A context-aware application offering map orientation

ALEJANDRO ARCOS

KTH Information and
Communication Technology

Master of Science Thesis
Stockholm, Sweden 2010

TRITA-ICT-EX-2010:36

A context-aware application offering map orientation

Alejandro Arcos
alej@kth.se

11-03-2010

Masters Thesis

Examiner and supervisor:
Professor Gerald Q. Maguire Jr.

Abstract

In this thesis context refers to information about the environment (the user or entity's surroundings) that can influence and determine the behavior of a computing system. Context-awareness means that the computer can adapt to the situation in which it is working. Context is a key issue in mobile computing, especially with handheld devices (such as PDAs and mobile phones), due to the fact that they can be used while on the move; hence the environment around them can change. The environment of a static device may also change and require the device to adapt. Applications and systems that exploit context by both sensing and reacting to their environment are called context-aware applications. Devices that are context-aware are able to perceive stimuli and react accordingly, with minimal interaction with the user.

Providing context-aware services to users of mobile devices via context-aware applications is becoming an important and significant factor in the market and is a developing industry. In this thesis we analyze and develop an application that exploits context to provide a service that improves the interaction between humans and a computer. The thesis begins with a study of what types of sensors are available to provide information about the device's context. This is followed by the design of an appropriate way of using the selected sensor (e-compass) to provide a means of adapting a service to the user's and device's context. The focus is every day activities of a student at a university - specifically finding the location of a meeting room for a seminar; however, similar scenarios exist for other types of users.

After determining that it was feasible to add a e-compass as a sensor to an existing personal digital assistant and to provide a map to the mobile user, the focus of the thesis shifted to an examination of the performance of the adaptation of the map as the user moved the device. Initially it required excessive time to render the map on the device, thus as the user moved the device the map was not updated quickly enough for the user to know their correct orientation with respect to the map. Therefore the thesis project examined how this performance could be improved sufficiently that the rendering would keep up with the change in orientation of the device. This investigation lead to a shift from server based rendering of the map as an image, followed by the transfer of the image to the device for display; to a sending a scalable vector graphics version of the map to the device for local rendering. While initially this was expected to be much faster than transferring an image for an actual map of the building where this work was taking place the local rendering was actually slower. This subsequently lead to server based pruning of the irrelevant details from the map, then a transfer of the relevant portion of the map to the device, followed by local rendering.

Sammanfattning

I den här avhandlingen hänvisar 'context' till information om miljön (i användarens eller enhetens omgivning) som kan bestämma och påverka beteendet hos ett datorsystem. Contex-awareness innebär att datorn kan anpassa sig till den situation där den arbetar. Context är en central fråga för mobila enheter, speciellt för handhållna enheter (t.ex. handdatorer och mobiltelefoner), på grund av att de kan användas på resande fot där omgivningen hela tiden förändras. Omgivningen för en statisk enhet kan också förändras och kräver att enheten kan anpassa sig. Applikationer och system som utnyttjar context genom att både känna av och reagera på sin omgivning kallas context-aware applications. Enheter som är kontextmedvetna kan uppfatta stimuli och reagera på den med minimal användarinteraktion.

Att tillhandahålla kontextmedvetna tjänster till användare av mobila enheter via kontextmedvetna applikationer blir en allt viktigare och betydelsefullare faktor på marknaden och är en växande industri. I den här avhandlingen analyserar och utvecklar vi ett program som utnyttjar kontext för att tillhandahålla en tjänst som förbättrar samspelet mellan människa och dator. Avhandlingen inleds med en undersökning av vilka typer av sensorer som finns tillgängliga för att tillhandahålla information om enhetens kontext. Detta följs av en design för att på lämpligaste sätt använda den valda sensorn (e-kompass) för att tillhandahålla ett sätt att anpassa en tjänst till användaren och enhetens kontext. Fokus är vardagsaktivitieter för en student vid ett universitet - särskilt att hitta till ett konferensrum för ett seminarium, liknande scenarier finns även för andra typer av användare.

Efter att ha fastställt att det var möjligt att koppla en sensor, i form av en e-kompass, till en befintlig personal digital assistant samt att visa en karta för användaren, flyttades fokus för avhandlingen till en undersökning om tjänstens prestanda när användaren flyttade enheten. Initialt krävde enheten väldigt lång tid att rendera kartan och när enheten flyttades uppdaterades kartan inte tillräckligt snabbt för att användaren skulle veta sin riktning i relation till kartan. Därför undersöktes hur prestandan kunde förbättras så att enheten skulle kunna hänga med bättre när enhetens riktning förändrades. Undersökningen ledde till att istället för att rendera en bild på servern och sedan skicka till enheten, skapa en vektorbaserad bild på servern, skicka till enheten och rendera lokalt. Även om detta initialt förväntades vara mycket snabbare än att överföra en bild av en verklig karta var den lokala renderingen faktiskt ännu långsammare. Detta ledde till en serverbaserad gallring av ovidkommande kartdetaljer samt beskärning innan kartan fördes över till enheten och renderades lokalt.

Acknowledgements

Firstly I would like to thank my academic advisor and examiner Professor G. Q. Maguire Jr. for his constructive advice, and always fast and profitable feedback when I delivered my problems to him. His continuous support and guidance in various stages of the project was essential in the development of this thesis.

I would also like to thank some students that helped me to complete the project and made my stay at the Wireless@KTH lab an enjoyable experience: Alexander Riedel, who helped me in the first stages of the thesis; Yunlong Huang, who was a nice company during long time in the laboratory and gave me great advices for programming the WASA board; Petter Lindgren, who made a very satisfactory translation of the abstract in Swedish; and Carlos Herranz with whom I shared a lot of coffee and chocolate breaks, and made my time in the laboratory more entertaining.

Table of Contents

Table of Figures

List of Tables

List of Acronyms and Abbreviations

ADC	Analog to Digital Converter
API	Aplication Programming Interface
CPU	Central Processing Unit
CTS	Clear To Send
DCD	Data Carrier Detect
DCE	Data Circuit-terminating Equipment
DNS	Domain Name System
DSR	Data Set Ready
DTE	Data Terminal Equipment
DTR	Data Terminal Ready
FTDI	Future Technology Devices International
GPIO	General Purpose Input/Output
GPL	General Public License
GPS	Global Positioning System
GSM	Global System for Mobile communications
HTTP	Hypertext Transfer Protocol
I/O	In/Out
ID	Identifier
IDE	Integrated Development Environment
IETF	Internet Engineering Task Force
IP	Internet Protocol
IR	Infrared radiation
IRDA	Infrared Data Association
LDR	Light Dependent Resistor
MMUSIC	Multiparty Multimedia Session Control
NGDC	National Geophysical Data Center
OS	Operative System
PC	Personal Computer

PDA	Personal Digital Assistant
PHP	Hypertext Pre-processor
PIR	Passive Infrared Sensors
PNG	Portable Network Graphics
PSTN	Public Switched Telephone Network
RTS	Request To Send
RxD	Receive Data
SCL	Signal Conditioning Unit
SIP	Session Initiation Protocol
SMTP	Simple Mail Transport Protocol
SOAP	Simple Object Acces Protocol
SVG	Scalable Vector Graphics
TxD	Transmit Data
UART	Universal Asynchronous Receiver-Transmitter
UA	User Agent
URI	Universal Resource Identifier
URL	Uniform Resource Locator
USB	Universal Serial Bus
UV	Ultraviolet Light
W3C	World Wide Web Consortium
WLAN	Wireless Local Area Network
XML	Extensible Markup Language

1 Introduction

Modern sensing technology and research is fostering the development of context-aware services for different kinds of environments and purposes. Using context potentially improves human-computer interaction (i.e., interaction is easier and more comfortable for the users), thus sensing technologies have many important commercial opportunities.

For each type of sensor some value can be measured about the environment, we start by considering the range of possible and useful context-aware applications that might make use of the data. It is important to design application programming interfaces (APIs) to retrieve the information provided by these sensors, as suitable interfaces make it easier to develop applications and services based on the data from these sensors. The application programmer should not have to be concerned with data acquisition details or the details of the hardware. However, the application programmer might need to know details of the sensor's characteristics in order to take advantage of the properties of a particular sensor.

Additionally, the information provided by these sensors may be used by computers that are not directly attached to the sensor. In this case, we need to implement a sensor network or otherwise communicate the data from the different devices to applications that may make use of it. At the same time, it is necessary to protect the user's privacy and integrity, by only passing this data to the devices and applications that he or she wants to have access to this data at this time. There are many existing approaches to communicate information from sensors to other devices. In some of these approaches, modular devices with sensors communicate with servers. These servers in turn listen for client requests, in order to make the data supplied by the sensors available to other computers. This has the advantage that the sensor node can focus on sensing and does not need to process requests from various clients, hence it is quite scalable. In other approaches, the sensors make client requests (and pass the sensed information) to servers. These servers can compute some function over the data and send a response back to the client. This later approach might be used by sensors to do sensor synthesis over the data from many sensors or from different types of sensors. Depending upon the use that will be made of the sensor data, the developer should choose the most suitable approach to build their sensor enabled application.

The initial goal of this thesis project was to analyze and evaluate different types of sensors, suggest new context-aware applications, provide a suitable Application Programming Interface (API) to access these sensors that are used, and to design, implement, and evaluate an approach that would provide mobile users with interesting and useful content-aware services.

More specifically for this thesis project, we have at our disposal a highly sensitive magnetic field sensor, which can be used to measure the weak magnetic field of the earth, that can be used to build an electronic compass application. Given the device's location, for example, from GPS receiver, and information about orientation provided by such an

electronic compass, a service could be implemented that displays a map for the user's positioning properly oriented on the screen according to the current orientation of the device (See Figure 1). In order to provide such a map, the user request includes as parameters: orientation and position. This information could be included by adding this information to the header of an HTTP request, in the body of a request, or as part of the URL being requested. (See section 2.6 for examples and details.). The web server could generate a map with the appropriate orientation and send it in the response to the user. Alternatively the server could provide a map for this location, but the application could render it locally. Later in this thesis we will compare these two approaches.

Figure 1. Map positioning using orientation

Another example involves using an accelerometer to provide information concerning which way the screen of a device is oriented. This information can be communicated to a presence information system for example in a Session Initiation Protocol (SIP) architecture. If the device (for example, a Personal Digital Assistant - PDA) is turned upside-down (determined by measuring gravitational acceleration), this might indicate that the user is not available. When the device is turned face upwards again, the SIP presence server might be notified that this user is now available.

Other sensing technologies that measure light, temperature, presence, sound, image pixels, etc. can be used for different applications (see section 2.2 in page 5). This thesis begins by describing context-awareness, sensing, existing applications, and describes new

sensors and new applications.

The thesis is organized as follows. Chapter 2 will summarize the basic concepts (context-awareness, SIP, and web services) and technologies (sensors, WASA board, and magnetic field sensor) on which this thesis is based. Chapter 3 will present a description about interfacing sensors to a PDA, which covers the communication and powering from the PDA to the WASA board, and the specific interface for the magnetic field sensor. Chapter 4 analyze different approaches to implement an application that makes use of the data provided by the magnetic field sensor and displays orientated maps in a PDA. Finally chapter 5 contains some concluding remarks and presents some open paths left by this thesis that can be faced in the future.

2 Background

This chapter gives a description of the concepts needed to understand this thesis and next chapters, and an overview of the different technologies used.

2.1 Context-awareness

In computer science context-awareness refers to the idea that computers can both sense, and react, adapting to the situation in which they are working. The goal of context-awareness is to minimize the required user interaction by automatically responding to changes in the environment. As stated by Dey [1], context is "any information that can be used to characterize the situation of an entity." According to this definition, it is possible to consider any data collected by sensing as context, thus with advances in sensor technologies, the scope of context-aware computing increases.

The most common information used by context-aware systems is the user's location and position. These two sources of context information have been widely investigated. Albrecht Schmidt [2] in his dissertation "Ubiquitous Computing - Computing in Context" describes several different technologies for sensing location.

GPS is very popular and easy to use for determining location outdoors. The output of a GPS receiver is the position of the device and the local time. The accuracy of this approach depends on the number of satellites which are visible, the signal strength of the signal from these satellites, and the availability of other information that can be used to improve the time to first position report (for example, getting a rough location using another technology so that the search time can be reduced, getting an accurate time value from another source so that the synchronization with the pseudo-random spreading code used for the radio signals can be improved, differential updates transmitted by fixed GPS receivers located at known locations, phase differential receivers, ...). While GPS allows rather accurate positioning (ranging from 10m to sub-cm accuracy) outdoors, the penetration of GPS signals is very limited in many buildings and is also limited at high latitude (especially those with trees or buildings). Unfortunately, Stockholm is located at a high latitude and many of the applications that would like to use positioning occur indoors. In addition, a large portion of users spend much of their time in doors - so while GPS works well for vehicular based systems, it is not as suitable for human carried systems.

Another method is to listen for radio or TV transmissions, for example using the information provided by base stations of wide area cellular networks. Many applications simply use the cellular basestation's ID to estimate the location of the receiver. More sophisticated solutions use information about the range and link quality of the signal received from multiple cellular base stations. This approach of listening for transmitted signals and using this information to estimate the location of the receiver has been widely explored - with both simple and very sophisticated solutions. These solutions are often useful both indoors

and outdoors, because of the penetration of the signal into (and out of) buildings and because many transmitters exist. So in the locations where there are regularly people, there are frequently transmitters that can be heard.

Many researchers have used information from WLAN base stations (or WLAN equipped mobiles) to estimate the position of mobile WLAN equipped devices. These range from very sophisticated commercial systems (such as that by Ekahau [3]) to very simple systems that simply compute a signature based upon the transmitters that can be heard (possibly also considering their signal strengths). An example of such a simple location system is that described by Haruumi Shiode [4] in his masters thesis. He utilized measurements of WLAN signal strength from mobile devices as they moved down a corridor of a building. An interesting aspect of his solution was the use of a highly directional antenna directed along this corridor and the use of both the received signal strength and the choice of frequency used by a mobile - that is communicating with various access points in the building (as the mobile station will choose which base station, i.e., access point) it associates with as it changes position. This very simple system was able to estimate the user's position in this corridor to ~2 m (sufficient to know which doorway a user was in front of).

Although initially context-awareness was widely perceived to be primarily based upon using the user's location, there are many other measurable aspects of a user's context that can provide useful information to develop context-aware applications, such as light, vision, temperature, humidity, air pressure, sound, motion, acceleration, orientation, human touch…, etc. For example, one of the earliest context-aware platforms with multiple sensors was the first SmartBadge at KTH. This device used temperature, humidity, light level, and orientation. (See [5] for further details.)

2.2 Sensors

In this section we present a description of sensing technologies that exist today and could be useful for context-awareness. Many of these context sources have been examined before (such as in [2] and [5]).

It is important to say that there have been great improvements made with respect to physical size and weight, power consumption, processing requirements, interfacing options, reliability, robustness, and price for many of these sensor technologies. Due to these developments, we believe that a variety of sensors are now useful (although they previously were not as practical or feasible) and it is now worth the effort to re-consider them for context-aware systems.

2.2.1 Light

Optical sensors can be used to gain information about the light intensity, reflection, and wavelength or the type of light (such as sunlight or artificial light). The last of these is possible by using two or more light sensors by exploiting their differences in sensitivity

(response) to specific wavelengths or an specific spectral range. Sensors that measure the colour temperature of the light (wavelength) or UV (ultraviolet light) sensors can be used to determine if a device is indoors or outdoors.

Light sensors can have very low cost and are a rich source of information. The most common light sensor is the photoresistor or Light Dependent Resistor (LDR), whose resistance decreases with increasing incident light intensity. The energy consumption of a LDR is low, and they are simple to interface to a microcontroller.

Additionally, many devices are equipped with an IR transmitter and receiver, for example for use as a communication link following the IRDA standards [7]. Such a receiver can be used to listen for an IRDA beacon, such as the HP Laboratories CoolTown beacons (for an example of this see [8]).

2.2.2 Audio

Microphones and amplifiers can be used to capture audio information in an environment. The audio can be restricted to certain frequencies, by using microphones with different frequency spectra. Today a very large number of devices are already equipped with a microphone and an analog to digital converter - often a converter with a sampling rate of 44.1 or 48KHz. This enables very wide band audio signal processing (limited largely by the microphone(s) used).

It is possible to get information about the location of an audio source by employing multiple microphones that are spatially distributed (see for example the microphone arrays - such as those supported under Microsoft's Vista operating system). Identifying context can also take advantage of detecting specific sound sources, such as background sounds (trains, clocks, fans, ...), the voice of a specific person or animal, the acoustics of a room (due to the differences in the way that sound is reflected or not from the walls, furniture, people, etc.), ...

2.2.3 Movement and acceleration

As people and objects generally do not change their whereabouts without movement and acceleration, information about acceleration is a very helpful source of context information. This source of data is even more important when considering light weight and ultra-mobile hand-held devices, because changes in the way they are handled can provide functional information. Human interaction with artifacts usually involves some movement, hence acceleration.

There are diverse ways of sensing movement, with different cost and accuracy. As a representative example, consider: motion switches, accelerometers, and gyroscopes.

Motion switches are a simple mechanism for detecting movement. Any movement that results in a change of the position of the conductive element in the switch will cause a change in its state (closed or open). These sensors are often very useful as they offer a simple means

for waking up microcontrollers when they are in an energy saving (sleep) mode.

Accelerometers are usually integrated micro-machine devices combined with driving electronics in an integrated circuit. These sensors are easy to connect to a microcontroller, and their power consumption is low and their size is rather small. Accelerometers provide information on the acceleration in one (single axis models), two or three (multi-axis models) directions. Multi-axis models can detect magnitude and direction of the acceleration as a vector quantity. They make it possible to detect the inclination of a device, by determining the component of the acceleration caused by gravity. If the original position, velocity, and orientation of a device are known, it may be possible to find out its current position by estimating its movements based upon integrating the acceleration sensed. However, the magnitude of acceleration is less useful if there is no knowledge of the device's orientation. For examples, of devices using accelerometers see [9], [10], [11], [12], [13], and [14].

Gyroscopes are used when angular velocity is of interest. These devices are generally more expensive, bigger in size, and have greater power consumption than accelerometers.

2.2.4 Magnetic Field and Orientation

There are different types of sensors that are able to detect magnetic fields. Some of them are used to perceive the earth's magnetic field, whereas others are designed to detect the proximity of a generated magnetic field. Sensors that use information about the earth's magnetic field can be used as an electronic compass, i.e., its output can be used to know which direction is north. More about this in section 2.5 on page 12.

2.2.5 Proximity, Touch and User Interaction

Sensing that a user interacts with a device can be implemented in a variety of ways and using several sensor technologies. Detecting that a person is close to a device can indicate that a user interaction is likely to happen. Capacitive sensors are a simple and frequently used solution for this. They can output a digital signal when a limit is crossed or their analog output can be used to provide a much more complex control signal (for example as in the musical instrument: theremin). Humidity sensors can also be used to detect proximity, as humidity increases when a human approaches. However, the rise in humidity is small, hence high sensitivity sensors are required.

For capturing direct user touch, conductive surfaces are used, which exploit skin conductance. Another option is to use force sensitive resistors, which change their resistance according to the force applied to them.

Information about touch and proximity enables us to know when the device is being used or to anticipate that the device will be operated in the near future. If devices are only operational when in the user's hand (or being manipulated by the user's hand), these sensors can be used to significantly reduce power consumption.

2.2.6 Temperature, Humidity, and Air Pressure

Sensing temperature is rather simple and cheap using temperature dependent resistors. Knowing the temperature can be helpful to determine in which environment a device is being used. For example, given a temperature reading of +3ºC, it is high likely that the device is outdoors at that moment, but a temperature reading of +20ºC does not indicate if the device is indoors or outdoors, because this temperature could be common to both scenarios. However, a temperature measurement at the device could be compared with temperature measurements of the outdoors and various locations indoors. Temperature measurements are generally part of the weather news and are often available on-line, while there are an increasing number of networked environmental sensors in buildings (for climate control) and lots of sensors transmit their readings over a wireless link. See for example the TFA-Dostmann weather stations [31] and other 433 MHz sensors (see section 7.2 of [15]).

By using high accuracy measurements, we are able to find transitions between situations, clues about usage patterns, and changes in the surrounding environment. Often the information about a transition may be more useful than an exact measurement of the temperature or humidity. For example, knowing that there is a change may indicate that the user has changed their location or their activity.

Sensors for humidity can be used for measuring weather conditions, detecting changes in the environment, and, as described in section 2.2.5, perceiving presence.

Air pressure gives an indication about altitude and can be used as a barometer. Changes in air pressure can indicate actions such as closing a door in a room or a vehicle, going up or down in an elevator (or stairs), or driving through a tunnel.

2.2.7 Motion Detection

Detecting motion of subjects within a certain space can be obtained using different technologies. Two of the major methods are infrared sensors and cameras.

A common way of motion sensing is the use of Passive Infrared Sensors (PIR). PIR sensors perceive changes of heat flow in the environment and can detect people or animals moving in the detector region of the sensor. Some of them can offer binary information about if someone entered or left the detector region. They can have different observation angles and detection distances. The use of PIRs for determining the direction of the user's movement (in particular whether a user is entering or leaving a room is described in [16] and [17]).

Motion sensing is specially useful with low cost stationary sensors, for example to detect if someone enters or leaves a room. In mobile devices, motion sensing is more difficult because when the device is moving, it is hard to determine if something around is in motion or it is the device itself that is moving. However, changes in relative positions between objects may still be a useful source of information.

Video analysis for detecting motion is more powerful than PIR sensors, but by far more

expensive and complex to implement. As described in the thesis by Daniel Hübinette we will not further consider video analysis because of the prohibitions on public cameras at KTH (and in most working environments in Sweden).

2.2.8 Bio-Sensors

Bio-sensors are used to measure signals from living creatures in different ways. Their use is common in medical applications, but some sensors are integrated in mobile devices for everyday use.

There are various sensors available to measure pulse or heart rate at different body points. This indicates how calm, nervous, or exhausted someone is, and is especially helpful in applications that are designed to be used during practicing for a sport. See for example the use of remote monitoring of an athlete by a coach in [18].

There exists other sensors that measure skin resistance to provide an indication of the tension and excitement of the user, for example as lie-detectors. Muscle tension can also be measured and this information can be used in bio-feedback systems. Finally, there are also sensors to acquire blood pressure values, and measure the activation pulse of the heart, both of them with evident medical applications.

2.3 Session Initiation Protocol

The Session Initiation Protocol (SIP) is a signaling protocol, used for controlling (to establish, modify, and end) a multimedia communication session, such as voice or video calls, over the Internet Protocol (IP). SIP is a text-encoded protocol and is based on elements from the Hyper Text Transport Protocol (HTTP) and Simple Mail Transport Protocol (SMTP), which are used for Web browsing and for e-mail on the Internet (respectively). It was initially developed by IETF's Multiparty Multimedia Session Control (MMUSIC) working group, but evolved to have its own SIP working group within the IETF. The protocol was designed to provide call setup and signaling in an IP based network, but it also has other functions, such as presence notification and short messaging [6]. The differences between SIP and the traditional Public Switched Telephone Network (PSTN) are large because SIP end-points have significant computing capabilities that can provide multiple services, while PSTN systems generally have a fixed quality as the end-point do not have any significant processing capabilities.

There are three main elements in a SIP network: User Agents, Servers, and Location servers. These SIP user entities are introduced in Bemnet Tesfaye Merha's master thesis [8]. Here we will only give a brief introduction, please see his thesis for additional details of a SIP network.

SIP users are addressable entities that participate in SIP sessions. Users are indentified with a Universal Resource Identifier (URI), with a format similar to an e-mail address. A SIP

URI generally has the form *sip:name@domain:port* where *name* is the name of the user; *domain* is the fully qualified domain name of the user's proxy server, and *port* is the port number where the proxy server is listening for a connection. It is also possible to specify phone numbers that can be reached via a gateway between the SIP network and the PSTN. For example, the URI of the form *sip:+46-700-680-137@gateway.com; user=phone* may refer to a PSTN phone number.

SIP user agents (UAs) are end-point devices in a SIP network used for sending and receiving messages. They originate SIP requests to establish media sessions and send & receive media. A user agent can be either a hardware device (for example, a SIP phone), or a SIP client software running on a PC or handheld device. Alternatively SIP user agents can also be a gateway to another network, such as a PSTN gateway.

Each user agent must have at least one valid IP address, and should be able to resolve a domain name (by utilizing a DNS server or other means). SIP users, with their corresponding fully qualified URI, are associated with a user agent by registering with a SIP registrar server. A user agent will be able to receive an incoming invitation once its current SIP user agent and its location are known by the registrar server.

SIP user agents can communicate in a peer-to-peer way, but its convenient to use another network element called a SIP proxy to help user agents to initiate SIP sessions. A SIP proxy receives SIP requests from a user agent or another proxy and forwards the request to another SIP entity. SIP proxies also provide a number of security functions and support the definition and enforcement of user preferences.

A redirect server receives a request from a user agent or proxy and returns a redirection response (3xx), indicating where the request should be retried. The main reason for utilizing these servers is to reduce the load on proxy servers, which are also responsible for routing tasks.

As it has been said before, a SIP user agent must register its contact information with a registrar server in order to receive incoming invitations. The registrar server uses a location server to store the information related to a SIP user. Location servers utilize a database to store each user's contact information, IP addresses, and port numbers. These location servers are not accessed directly by user agents, but registrar servers and proxies utilize this information to provide services to user agents.

2.4 WASA Board

The WASA board is a circuit board designed and developed for teaching by Professor Mark T. Smith of KTH for his Sensor Based Systems course. A bachelors thesis by Thor Hådén [15] earlier described the WASA board. This board already has some on-board sensors and a number of General Purpose Input Output (GPIO) ports. On-board sensors integrated with the board are a temperature sensor, a LDR light sensor, and a 3-axis accelerometer. Most

other sensors can be connected to the GPIO ports or to the analog input or output ports (that are internally connected to an analog to digital or digital to analog converter respectively).

The WASA board communicates with a computer through a USB interface, emulating a serial port. They were designed this way because most students' laptops do not have serial interface, but do have a USB interface. In addition, USB provides power, thus there is no need for a separate power supply for the board. A user can directly send and receive data from the board using a terminal program such as Hyperterminal on Windows. There are also APIs for using serial ports in most programming languages. The software that is running in the WASA board accepts AT-commands for communication, which are based on a system created by Hayes to communicate with modems and dialers, where all commands begin with the string “AT”. These letters come from the word "attention", which is the first command that the computer sends to a modem/dialer in order to get its attention. The characters "AT" allow the modem/dialer to learn the proper data rate and to synchronize with the computer's serial interface. AT style commands have been used for the WASA board as they are straight forward to generate and decode, and because it is easy to add new commands to the protocol. In the case of the WASA board there exists a small program running in a microcontroller on the board that decodes the AT command to learn what sensor should be sampled and returns the sensor's current value as output.

A user, or computer program, writes a text string to the serial port's output buffers, which is transmitted to the board through the computer's USB interface. The program in the microcontroller on the board will receive the string from the computer, decode it, and execute some function according to the command string. If the command requests a reading from a GPIO port or sensor, the WASA board collects the value and writes a reply to its output buffer, which is transmitted back to the computer. To read the reply, the computer simply reads from its input buffer. Thor Håden describes in his thesis how he add a new 433 MHz radio transmitter to the WASA board and how he added a new command to the board to continuously digitize the output of this receiver. See [15] for details.

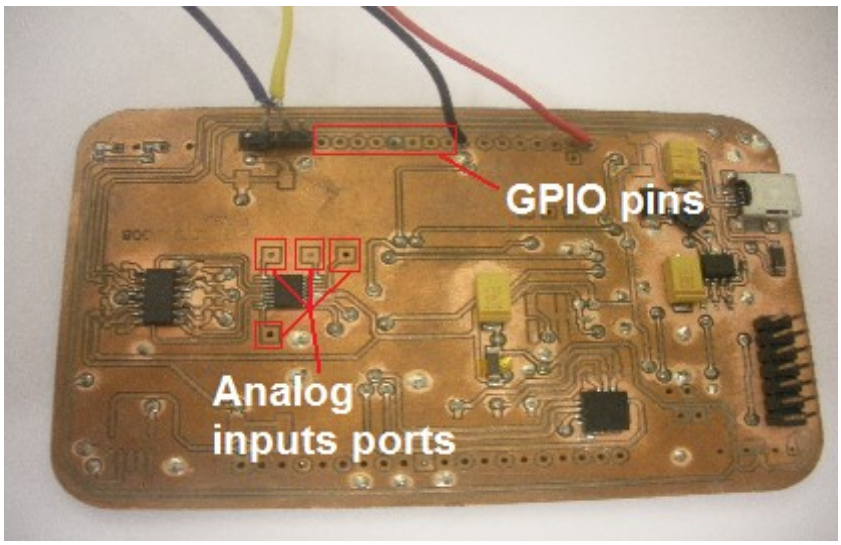


Figure 2. WASA board

2.5 Magnetic Field Sensor

The KMZ52 from NPX (formerly Philips Semiconductors) is a magnetic field sensor, designed for compass applications. This sensor relies on the magnetoresistive effect and provides the required sensitivity and linearity to measure the weak magnetic field of the earth. It implements a two-dimensional field sensor (note that the KMZ51 is single axis magnetic field sensor), as this is required for a compass.

The earth's magnetic field lines point from the earth's south pole to its north pole. These lines are perpendicular to the earth's surface at the poles and parallel at the equator. It is important to point out that the magnetic poles do not coincide with the geographical poles, which are defined by the earth's axis of rotation. The angle between the magnetic and the axis of rotation is about 11.5º. Thus we differentiate between "true North" (the North pole of the axis of rotation and point at 90 degrees north latitude) and "magnetic North" (see figure 3).

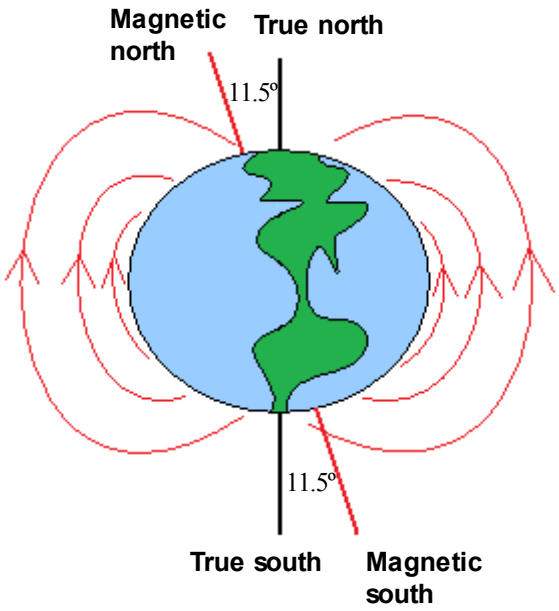


Figure 3. Earth's magnetic field

In Figure 4, vector **He** is the earth's magnetic field vector at some point on the earth. Some other quantities that are of importance for a compass are also defined. In this illustration, the x- and y-coordinates are parallel to the earth's surface, and the z-coordinate points vertically downwards (into the earth). Note that this is a righthanded coordinate system.

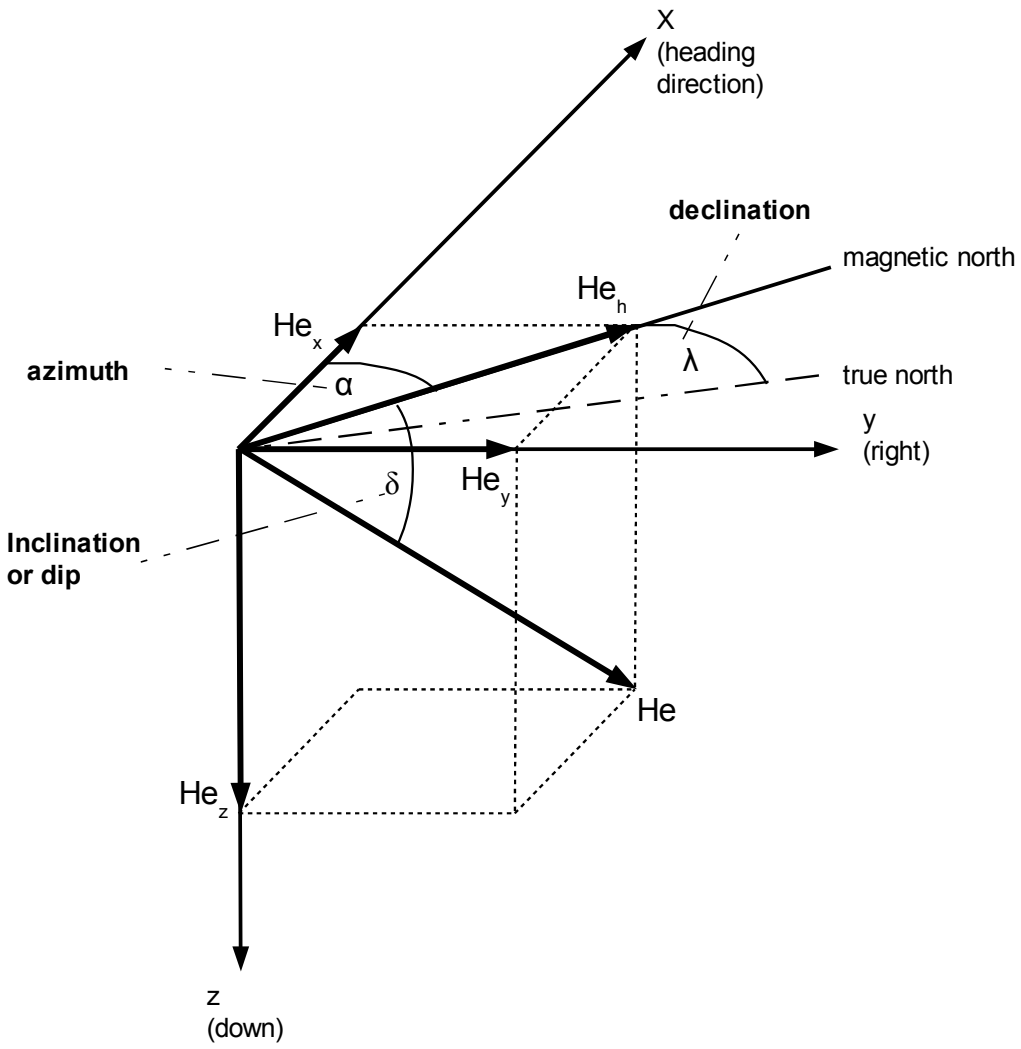


Figure 4. Earth magnetic field as a vector quantity

The angle between magnetic north and the heading direction is called azimuth and is represented by **α**. Magnetic north is the direction of $\mathbf{He_h}$, the earth's field component perpendicular to gravity. Azimuth is the desired output of a compass, and is defined by:

$$\alpha = \arctan\frac{He_y}{He_x}$$

Inclination δ is the angle between the earth's field vector and the horizontal plane. The inclination varies with the actual location on earth. If a compass is tilted, the inclination has to be considered.

Declination λ is the angle between geographic north and magnetic north.

The main building blocks of an electronic compass are two sensor elements for measuring the x- and the y-components of the earth field in the horizontal plane, a signal conditioning unit, and a direction determination unit.

2.5.1 Magnetic Field Sensors

The task of a compass is to measure the azimuth, as stated before. Therefore, the strengths of two horizontal earth magnetic field components have to be measured: one in heading direction (He_x) and one sidewards (He_y). This requires two magnetic field sensors, both aligned parallel to the earth's surface, but rotated by 90 degrees with respect to each other.

2.5.2 Signal conditioning unit

The purpose of this unit is to deliver output voltages proportional to the field strengths $\mathbf{He}_x$ and $\mathbf{He}_y$ respectively. The signals delivered by the magnetic field sensors are very small, hence they have to be amplified. Optional features for high performance systems are temperature compensation of sensitivity and compensation for the error due to non-orthogonality between the sensors. To carry out these tasks, the signal conditioning unit has to control the set/reset and compensation coils of the sensors.

2.5.3 Direction determination unit

The function of this unit is to calculate the desired azimuth information from the measured field strengths $\mathbf{He}_x$ and $\mathbf{He}_y$. The measured azimuth has to be indicated on some external display or communicated to another electronic system.

2.5.4 Other Features

In most practical scenarios, other magnetic fields can cause interference with the earth's magnetic field, which could produce a significant measurement error. Fortunately, there are calibration methods to compensate for this error.

It is also possible to correct for the deviation between the magnetic north direction as measured by the compass and the true or geographic north. The value of declination varies with the position on earth and can be to the east or to the west (east declination means that the magnetic north indicated by the compass is east of true north). Declination also varies in time, hence for compensation only updated information about declination can be used. The correction can use information from a remote data server as a function of the user's geo-coordinates. Such data for locations world-wide can be found at the website of the U.S. National Geophysical Data Center (NGDC) [19].

The azimuth equation only yields the correct value if $\mathbf{He}_x$ and $\mathbf{He}_y$ are the earth's field components in the horizontal plane. Because of this, the basic compass must be held exactly horizontal to work properly. However, it is possible to electronically compensate for the error that appears when a compass is tilted.

2.6 Web Services

In some applications it may be necessary to make client requests to servers that compute some function over the data provided by sensors, and return a response back to the client. Web services support interoperable machine-to-machine interaction over a network and provide a simple means of implementing client-server communication. Web services are typically conveyed using HTTP and are frequently based upon Internet Application Programming Interfaces that can be accessed over a network (such as Internet), and executed on a remote system hosting the requested services.

For transmitting data in the body of an HTTP request, web services can use Extensible Markup Language (XML) messages that follow the Simple Object Access Protocol (SOAP) stardard. SOAP is an standard protocol that defines how two objects in different processes can communicate by exchanging XML data. For example, a SOAP message could be sent to a web service enabled web site (for example, a flights company database) with the parameters needed for a search. The site would then return an XML-formatted document with the resulting data.

This could be the SOAP message of the client:

```
<source lang="xml"> <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body> <getFlightInfo xmlns=" http://flightcompany.example.com"> <flightNumber>1234</flightNumber>
</getFlightInfo> </soap:Body>
</soap:Envelope> </source>
```

And the response of the server:

```
<source lang="xml"> <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body> <getFlightInfoResponse xmlns=" http://flightcompany.example.com">
<getFlightInfoResult> <flightTime>15.45</flightTime>
<flightNumber>1234</flightNumber> <availabeSeats>43</availabeSeats>
<departure>London</departure> <destination>Paris</destination> </getFlightInfoResult>
</getFlightInfoResponse>
</soap:Body> </soap:Envelope>
</source>
```

Even easier implementations of web services can be done if client adds the data parameters to the URL of an HTTP request. For example:

```
http://flightcompany.example.com?flightNumber=1234
```

Web services can be a useful tool to implement functions that use information captured by sensors, but where the service cannot be executed in the device to which the sensors are attached. An extreme example of this is the emerging netbooks. These are portable devices that locally only have a browser and OS, as all other processing is done somewhere else in the network.

3 Sensor interfacing

This chapter will describe the design and implementation of the interface that connects the sensor to the mobile device. The interface is designed in terms of data communication and powering. The technologies used are also described.

3.1 Serial connection between PDA and WASA board

As has been said in section 2.4, the WASA board can communicate with a computer through a USB interface, emulating a serial port. The MSP430F2618 microcontroller on the WASA board has two Universal Asynchronous Receiver-Transmitter (UART) interfaces, each one with one pin to transmit output data and another to receive input data. To provide the USB interface to the board, it is necessary to convert the USB data from the computer, to serial UART data to the microcontroller. The WASA board integrates a FTDI Chip FT232R ([20]) for this purpose.

This thesis project is focused on handheld devices that collect information from sensors. Unfortunately, few of these devices offer a USB master interface, as they do not have enough power to supply the current required by the USB standard to attached devices. Some recent devices support USB To-Go, a USB interface standard that allows devices to be a USB master or slave, with the master providing much more limited power than a traditional USB master. Unfortunately, the HP iPAQ 5550 PDA that we will use for our experiments has only a *slave* USB interface and a serial port. We can use the iPAQ's serial interface to directly exchange data between the PDA and the microcontroller (i.e., we do not need to use the FT232R USB to serial interface chip). However, **not** using the USB interface raises another problem: How can we power the WASA board? If the power consumption of the WASA board and sensors is low enough, then we can power the board via the serial port. While the WASA board was designed to operate on very little power, is the available power enough in practice to power the WASA board? We will examine this issue in the following sections. However, first we have to address the basic issues required for the PDA to communicate with the WASA board.

3.1.1 Serial data exchange

The MSP430F2618 microcontroller in the WASA board has 4 general-purpose I/O pins that provide two UART interfaces: UART0 and UART1. The transmit and receive pins of UART0 are already connected to the USB-to-serial converter (i.e., FT232R chip), hence to enable serial communication between the PDA and the microcontroller the easiest approach is to use the UART1 pins. These two pins can easily be accessed on the board, as they are brought out to connectors on one side of the board (see Figure 7).

To enable the use of this second UART some changes were made in the software that is running in the microcontroller of the board[1]. These changes are required to set up the UART1

1 This code was provided by Prof. Smith on his course web site.

interface (setting the serial clock's frequency and modulation), define an interrupt handler that handles the reception of data by the UART1, and add a function to write data through the transmit pin of the UART1. The code can be compiled specifically for the MSP430F2618 using the Texas Instruments' IAR Embedded Workbench IDE [21]. The necessary changes to the initial are shown in Appendix A.

However, to actually enable communication between the PDA and the microcontroller, it is necessary to shift the output of the MSP430F2618's GPIO pins to RS232 voltage levels. This level shifting can be made using the MAXIM (now Dallas Semiconductor) MAX3241 transceiver [22], which can run at 3.3V (the same voltage as the MSP430F2618 chip). The device requires five small 0.1 µF external capacitors that are used by the transciever to create a charge-pump. The necessary circuit for this level shifting is shown in detail in Figure 5. To prototype this level shifting an existing power supply and level shifting board that was developed for the earlier SmartBadge was used. The attachment to the level shifter is shown in Figure 6.

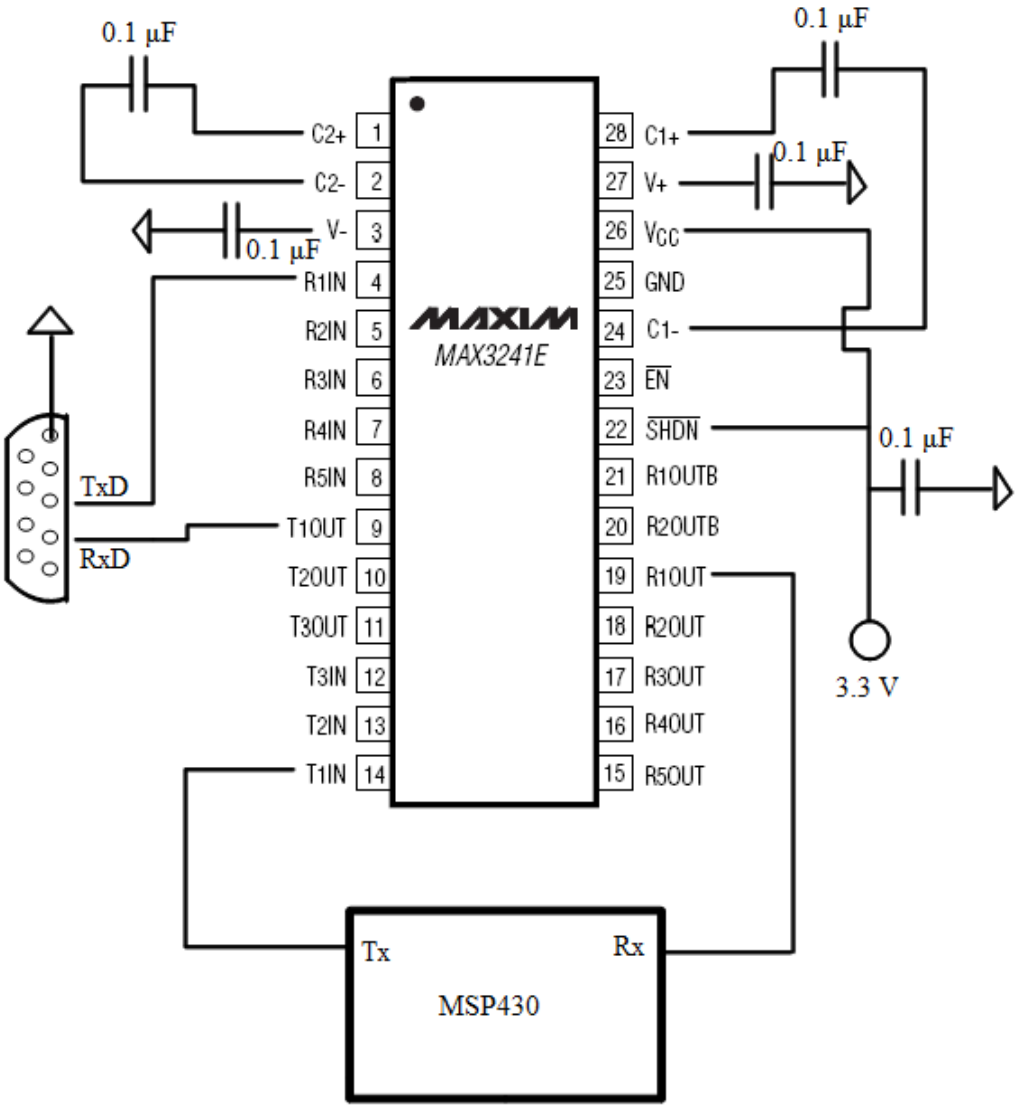


Figure 5. Level shifting schematics

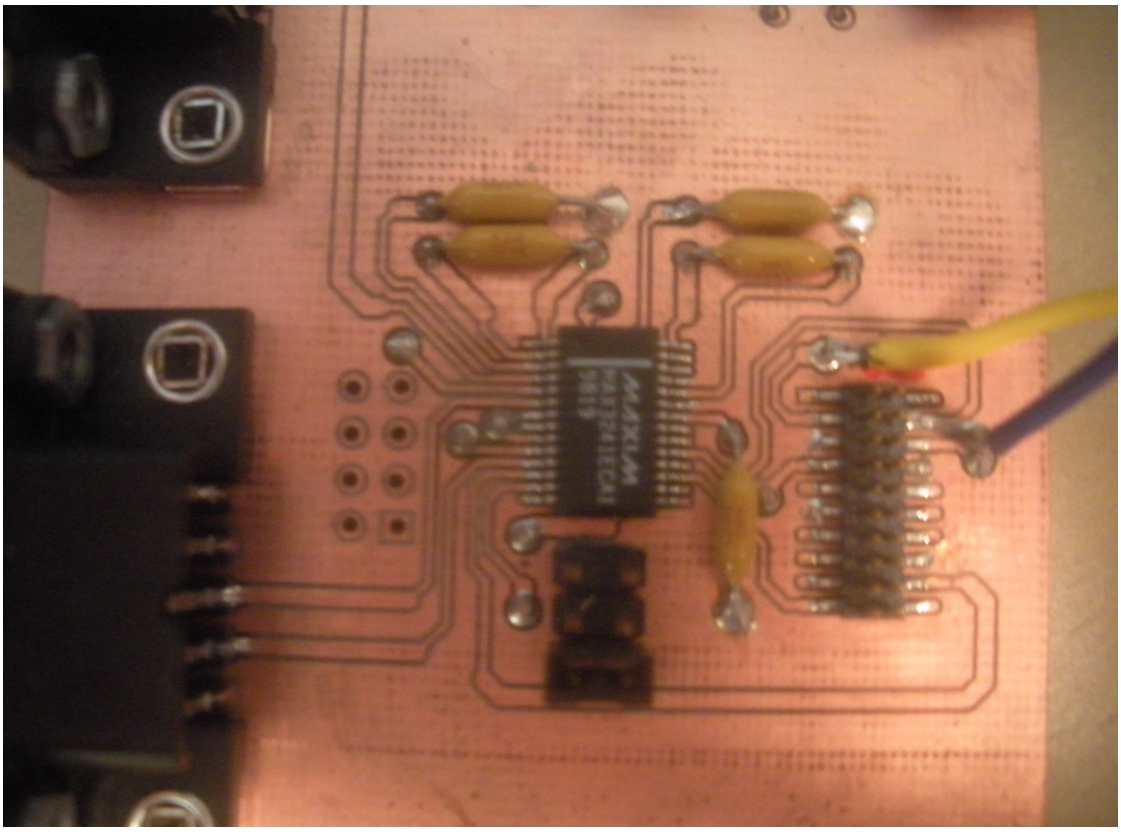


Figure 6. Level shifting board

Figure 7 shows the connections between the WASA board and the board with the MAX3241 and DB-9 (serial connector). The red wire provides 5V DC power from the WASA board to the level shifter and the black wire is a ground connection. The yellow wire connects the transmit pin of UART1 of the MSP430F2618 microcontroller with the T1IN pin of the MAX3241. The purple wire connects the receive pin of UART1 with the R1OUT pin of the MAX3241 chip.

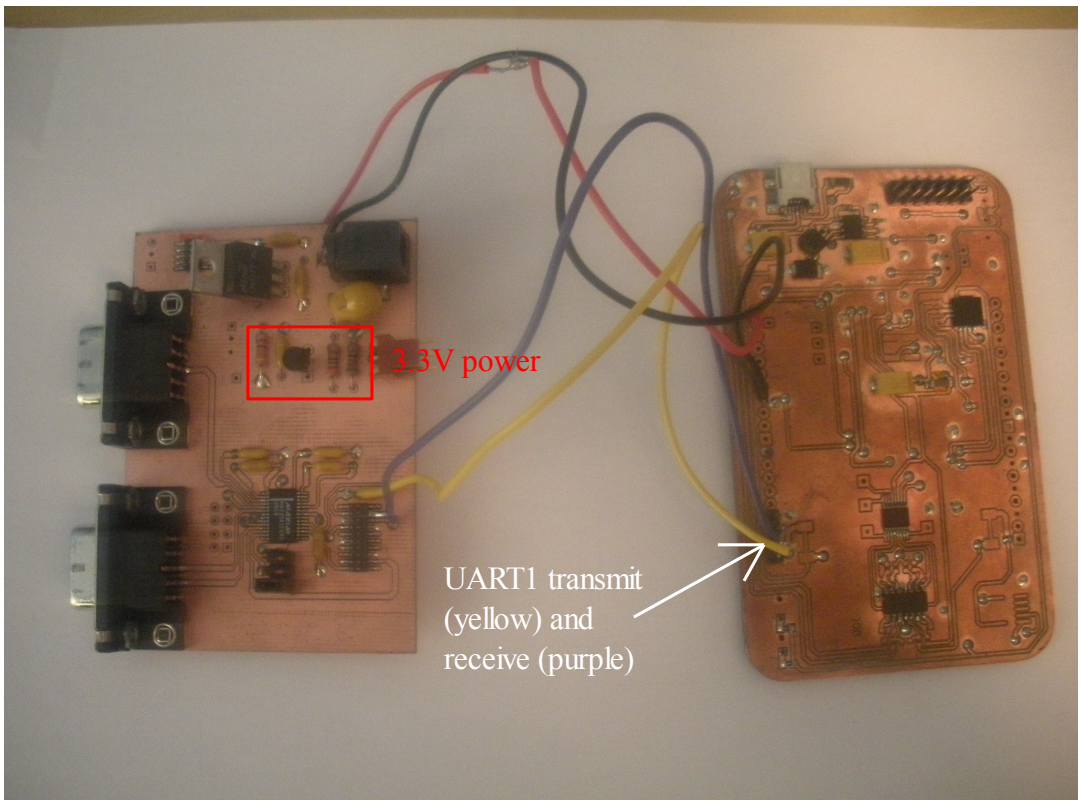


Figure 7. Connections between the level shifting board and the WASA board

Once these connections have been made, serial communication between the PDA and microcontroller is possible. However, the WASA board is still powered by means of the USB interface. The 3.3 V source that powers the MAX3241 could be provided by the WASA board (however, in the configuration shown in Figure 7, the 3.3V power for the MAX3241 is actually being provided by the small rectifier on the serial and power board).

Now that we have established a communication path between the PDA and the WASA board, the next problem is to determine whether is possible (or not) to supply the power needed using the PDA's serial port, rather than the Vcc (Common collector voltage) line of the USB interface. This will be addressed in the next section.

3.1.2 Power via the PDA's serial port

Powering a device using the serial port has been used in a number of applications. The most common example of this is a serial mouse (even if most serially attached mice have been replaced by mice with PS/2 or USB connections because these two interface can both provide power to the mouse). A serial mouse typically uses the DTR and RTS lines (which are used for handshaking, rather than for either transmitting or receiving user data) to provide power for a microcontroller circuit in the mouse. Before going into details of the DTR and RTS lines we described the RS-232 standard for serial interfaces.

The Electronics Industries Association (EIA) standard RS-232 is a standard for serial binary data signals between a Data Terminal Equipment (DTE) device and a Data Circuit-terminating Equipment (DCE) device. A DTE is the functional unit of a data station that serves as a data source or a data sink and provides for the data communication control function to be performed in accordance with a link protocol. A DCE is a device that sits between the DTE and a data transmission circuit. The original purpose of RS232 was to connect a terminal with a modem, hence the roles of DTE and DCE for the two ends of the connection. Although the application of RS-232 has extended far beyond this purpose, the original terms DTE and DCE are still used. Every device has to play the role of either a DTE or DCE.

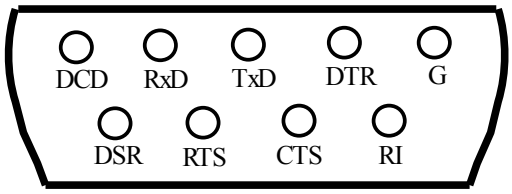
The standard specifies 20 different signal connections (hence older modems used a DB-25 connector). Today most devices use only a few of these signals, enabling smaller connectors to be used. For example, the 9 pin DE-9 connector is generally used for RS-232 communication. This is the connector that is provided by the iPAQ cradle. (See Figure 8) Hence this is the connector that we will consider for designing the first version of our serial port power and interface to the WASA board.

Figure 8. IPAQ cradle connector

Table 1 shows the different signals available on a DE-9 connector, their respective pin numbers, and whether their origin is in DTE or DCE device (the origin will drive the signal line with a voltage). Figure 9 shows the male pinout of this a DE-9 connector; while figure 8 shows the female connector.

Table 1. Serial signal description (pins are described from the view point of a DTE)

Name	Typical purpose	Abbreviation	Origin	Pin
Data Terminal Ready	Originated by the DTE to instruct the DCE to setup a connection. It means that the DTE is up and running and ready to communicate.	DTR	DTE	4
Data Carrier Detect	A signal sent from the DCE to its DTE to indicate that it has received a basic carrier signal from a remote DCE.	DCD	DCE	1
Data Set Ready	Originated by the DCE indicating that it is basically operating.	DSR	DCE	6
Ring Indicator	A signal from the DCE to the DTE that there is an incoming call. Only used on switched circuit connections.	RI	DCE	9
Request To Send	Originated by the DTE to initiate transmission by the DCE.	RTS	DTE	7
Clear To Send	Sent by the DCE as a reply on the RTS after a delay in ms, which gives the DCE enough time to energize their circuits and synchronize on basic modulation patterns.	CTS	DCE	8
Transmitted Data	Data sent by the DTE	TxD	DTE	3
Received Data	Data received by the DTE	RxD	DCE	2
Common Ground		G	common	5



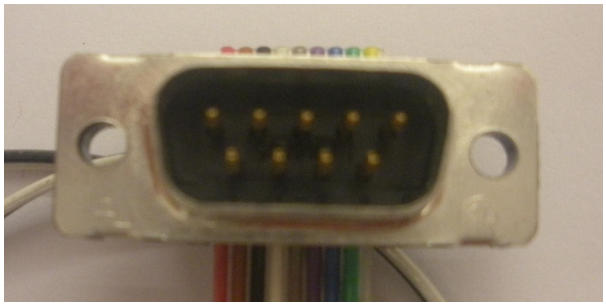


Figure 9. Male RS-232 pinout

Now that we are able to successfully exchange data between the iPAQ and the WASA board, the next step is to determine if it is possible to supply the necessary power to the

WASA using the PDA's serial port. We will do this by measuring the total power consumption of the WASA board and the level shifting board. This can be done by measuring the current flowing through the Vcc line when the WASA Board is connected to a PC by the USB interface. The measurements were made using a Velleman DVM92 multimeter, with the amperimeter function (black cable in COM plug, and red cable in 20A plug) at 2mA resolution. The value obtained experimentally for this current was 20.4 mA at 5 volts (i.e., the total power required is 102 mW power). Note that this power consumption includes the power required by the FTDI chip.

The next step was to determine how the power is shared between the WASA board and the level shifting board, by measuring the power carried by the +5V connection from the WASA board (the red wire in the figure 7). The value obtained experimentally was a current of 8.25 mA for the level shifting board (i.e., 41.25 mW). This implies that the WASA board takes 60.75 mW of power, leading to a current of 12.15 mA at 5 volts for its own supply.

Now that we know approximately how much power (expressed in terms of a combination of voltage and current) that is required for the WASA board and the level shifter, we have to measure the voltage and current output capabilities of the PDA's serial interface. However, first it is necessary to figure out if the PDA's serial interface is a DTE or a DCE, in order to determine from which pins it is possible to get power. In general, a DTE provides a voltage on TxD, RTS, and DTR pins; whereas a DCE provides voltage on RxD, CTS, DSR, and DCD. Therefore, whether the PDA is a DTE or a DCE can be determined by measuring voltages between the ground pin and the each of these pins. Non-zero voltage values were obtained for RxD, CTS, DSR, and DCD pins, hence the IPAQ PDA interface is a DCE. Specifically, the voltage value obtained was 5.64V for all four lines. This is in keeping with the fact that the RS-232 standard says that a female connector is used on a DCE and a male connector on a DTE (As can be seen in figure 8, the iPAQ cradle has a female connector).

Since the RxD pin carries out the task of transmitting data **from** the DCE, we have three lines (CTS, DSR, and DCD) that can be used to supply power. Table 2 shows the current driving capability obtained experimentally for each one of these pins (the measurements were made with the Velleman DVM92 multimeter, opening the DSR, CTS and DCD lines in a serial connection between the PDA and a PC, and using a resistor of 18 Ω):

Table 2. PDA's serial output current

Data Set Ready	1.21 mA
Clear To Send	1.27 mA
Data Carrier Detect	1.28 mA
Total	3.76 mA

From the measurements described earlier we know that 12.15 mA at 5V (60.75 mW) is consumed by the WASA board and 8.25 mA at 5V (41.25 mW) is used by the level shifting

board. From Table 2 we can compute that we can supply 3.76mA at 5.64 V (21.21 mW) of power using the three RS-232 pins from the PDA. As the amount of power available is not enough to power the boards as they are, it is necessary to analyze each one of the chips on both boards, and find out their power requirements, in order to understand if we can provide the power that is needed by the microcontroller, sensors, and level shifting circuits.

The WASA Board has a Texas Instruments MSP430F2618 microcontroller, which has a power consumption of 0.365 mA (in active mode running at 1 MHz at 2.2V), and can operate on a voltage ranging from 1.8V to 3.6V. This equates to a power requirement of 0.8 mW. It also has a FT232R USB to serial interface chip, which requires an operating supply current of 15 mA at a voltage ranging from 3.3V to 5.25V (i.e., ~50 mW). The WASA board includes a serial 8-bit ADC MAX1039 chip, which provides analog-to-digital conversion, and can be useful to interface to sensors. It needs a current of 0.350 mA at 188 kbps at a voltage between 2.7V and 3.6V (i.e., ~1 mW).

In the level shifting board, the MAX3241 transceiver is the only necessary chip. It requires a current of 0.3 mA at a supply voltage of 3.3V to 5V (i.e., ~1 mW).

Table 3 summarizes this information:

Table 3. Power requirements of the major integrated circuits that concern us

Chip	**Vcc – voltage (V)**	**Typical current (mA)**	**Power at 3.3 V (mW)**
MSP430F2618	1.8 to 3.6	0.365	~1.21
FT232R	3.3 to 5.25	15.000	~50.00
MAX1039 ADC	2.7 to 3.6	0.350	~1.15
MAX3241	3.3 to 5	0.300	~1.00

Given that the FT232 chip is unneeded in the case of a serial interface to the PDA and that it is a major power consumer, we find that the remaining power requirements (1.21 + 1.15 + 1 = 3.36 mW) can potentially be met by the power that can be supplied by the PDA's serial port (3.76 mA x 5.64 V = 21 mW). It might even be possible to power the required chips using a single pin (1.2 mA x 5.64 V = 6.7 mW) of the RS-232 interface.

The output voltage value obtained from the PDA's serial lines is 5.64V. The STMicroelectronics ST763A step-down switching regulator on the WASA board can be used to very efficiently convert the voltage from the PDA's serial port pins to the 3.3V needed by the microcontroller, the level shifter, and the ADC. The ST763A operates from 3.3V to 11V giving a fixed 3.3 output voltage. Given sufficient input current it can deliver up to 500 mA, which is more than enough to power the chips involved in the sensors that we are concerned with.

Now that we know that it may be feasible to power the WASA board via the serial interface we turn to the design of the electronic compass in section 3.2.

3.2 Interfacing to the NXP Semiconductors KMZ52

In section 2.5 an overview of the NXP Semiconductors KMZ52 magnetic field sensor was given. In this section we examine how this sensor can be interfaced to the WASA board in order to measure the device's orientation. In order to deliver output voltages proportional to the field strengths of the two components He_x and He_y that determine the magnetic north, we have to implement a Signal Conditioning Unit (SCU). Depending on the accuracy that is desired in the system, this SCU has to fulfill up to three requirements in order to eliminate the following error sources:

- **Offset voltages Vo_x, Vo_y at the SCU output.**
- **Sensitivity difference ΔS between x and y channel of the SCU.**
- **Non-orthogonality ß of sensors.**

For the offset compensation a technique called "flipping" can be used. This flipping is generated by applying alternately positive and negative current pulses to the set/reset coil of the sensor. When this is done repetitively, the desired output voltage will change polarity, appearing as the amplitude of the signal. However, the offset voltage will not change polarity, appearing as a measurable offset of the signal.

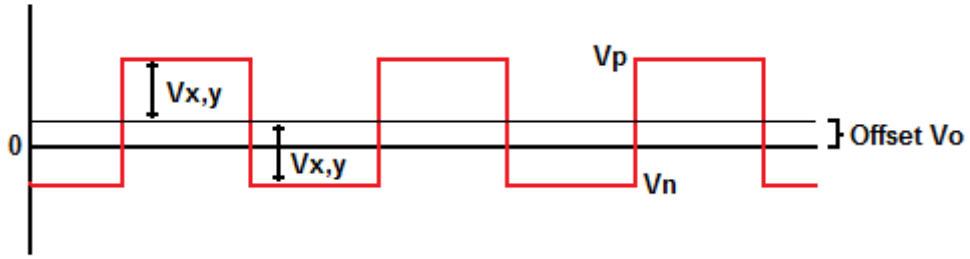


Figure 10. Offset compensation

As it is shown in Figure 10, the desired output voltages Vx,y can be calculated from the high voltage Vp and the low voltage Vn of the flipped and amplified sensor signal as:

$$Vx,y = 1/2 \cdot (Vp - Vn)$$

If the offset does not vary rapidly with time, then the processing load to measure the device's orientation can be reduced by reading Vp and Vn at greater intervals. Each time they are read, we calculate the offset voltage Vo, by computing $Vo = 1/2 \cdot (Vp + Vn)$. Once we have this estimate of Vo only one voltage of the flipped sensor signal has to be read, then we can calculate Vx,y as:

$$Vx, y = Vp - Vo$$

The sensitivity difference ΔS is due to tolerance and temperature drift of the sensor and the amplification circuit. The temperature drift can be compensated for by operating the sensors at the zero point of field strength, because it causes an output voltage of zero independently of temperature. Therefore, the earth field component at each sensor has to be compensated for by an opposing field of equal strength. This compensation field can be generated by supplying a current through a coil near the sensor that is integrated in the KMZ52 chip.

We can assume that the two magnetic field sensors are placed at an angle of exactly 90º, but in practice the displacement will deviate by an angle ϐ from the desired orthogonality due to mounting tolerances. This deviation causes an error in compass reading. The maximum error is approximately equal to ϐ. For the KMZ52 the specified maximum non-orthogonality is 2º, thus the maximum azimuth error is also 2º. If the compass is rotated in respect to the earth's field, then the phase shift between Vx and Vy is 90º±ϐ. Having determined ϐ, the error can be eliminated mathematically correcting Vy:

$$V_{yCorrected} = V_y / \cos\beta - V_x \cdot \tan\beta$$

There are two major ways of implementing the SCU, which both include pre-amplification of the voltage output, offset compensation by flipping, and temperature compensation of sensitivity by electro-magnetic feedback. One solution involves building all the blocks of the SCU using circuit components. The other solution, if a microcontroller is available, allow us to perform some of the tasks by software, simplifying the circuitry. By using a microcontroller, only the flip coil driver, pre-amps, and optionally the compensation coil drivers are required. Offset compensation can be implemented in software, according to the equations shown previously. Compensation coil drivers are unnecessary unless high precision is demanded. Even if high precision is needed, if a temperature sensor is available, then the error can be corrected by software. Further optional tasks, such as interference field calibration or true north calibration (see section 2.5.4) can also be performed by the microcontroller.

Figure 11 shows how the KMZ52 sensor can be interfaced to a microcontroller, without a compensation coil driver. In this case we only need the pre-amps and flip coil driver circuitry. The pinout of the sensor chip is shown in Table 4.

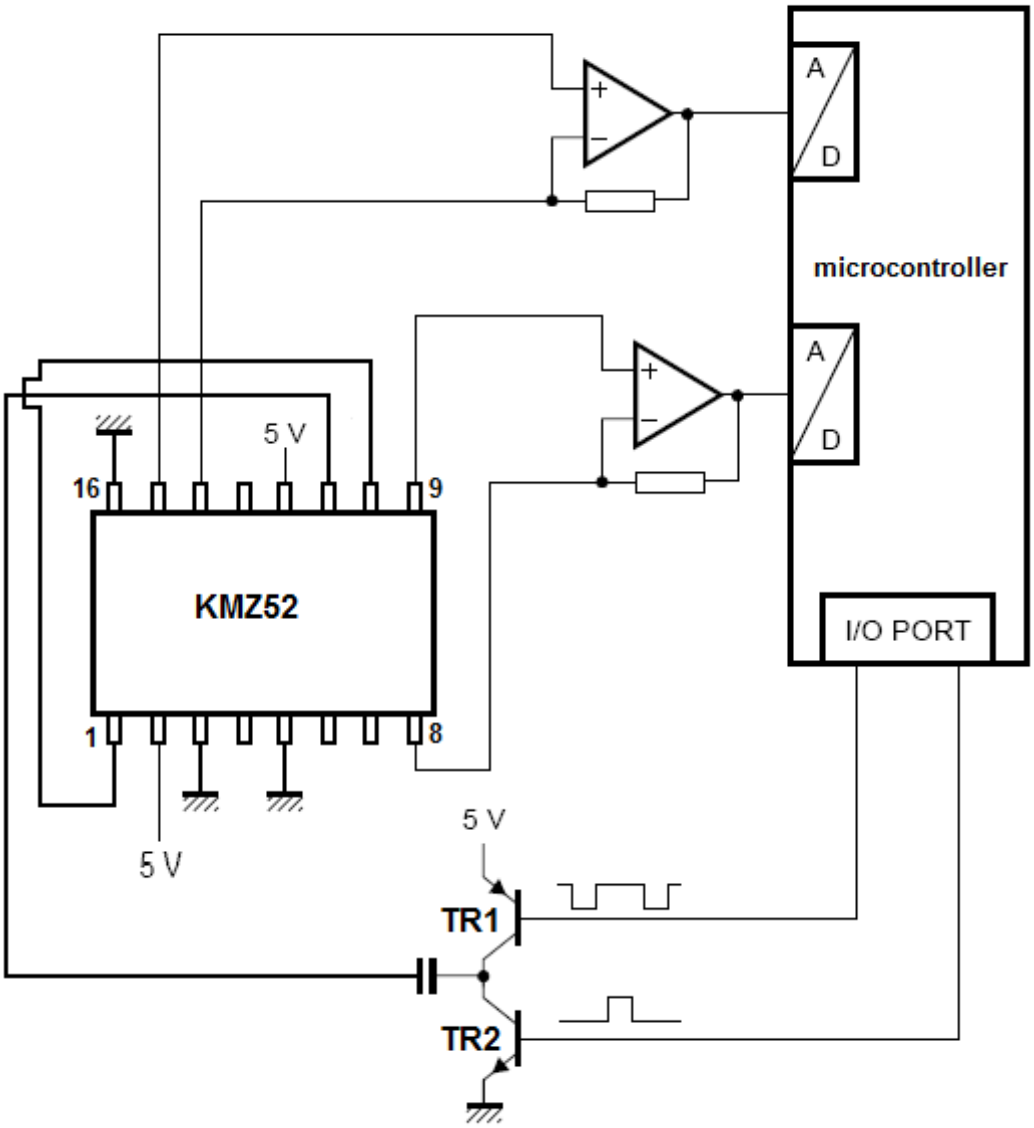


Figure 11. SCU with microcontroller

Table 4. KMZ52 pinout

Symbol	Pin	Description	Symbol	Pin	Description
+Iflip2	1	flip coil	+VO1	9	bridge output voltage
VCC2	2	bridge supply voltage	-Iflip1	10	flip coil
GND2	3	ground	+Iflip1	11	flip coil
+Icomp2	4	compensation coil	VCC1	12	bridge output voltage
GND1	5	ground	-Icomp2	13	compensation coil
+Icomp1	6	compensation coil	-VO2	14	bridge output voltage
-Icomp1	7	compensation coil	+VO2	15	bridge output voltage
-VO1	8	bridge output voltage	-Iflip2	16	flip coil

The flipping driver generates current pulses at some repetition frequency. This frequency is not critical (as long as it is sufficient to change the polarity of the output of the KMZ52). The choice of frequency will depend on the response time we want to achieve, with a faster response time increasing the average current consumption. According to the KMZ52 application note[1], a frequency of approximately 1 kHz is a reasonable compromise. The KMZ52 requires flip current pulses of typically ±1A for a duration of 3 µs (this implies an average power consumption of 15 mW). It is necessary to generate two signals from the microcontroller's I/O port. One of the signals produce a pulse that switches TR1 on. This charges the capacitor and a short positive pulse is passed to the flipping coil. The second signal forces TR2 to conduct, discharging the capacitor and providing a negative current pulse through the coil. The waveform of the signals depends on the flipping frequency chosen and the duration of the current pulses. Assuming a frequency of 1 kHz and a duration of 3 µs, the waveform should look like that shown in Figure 12:

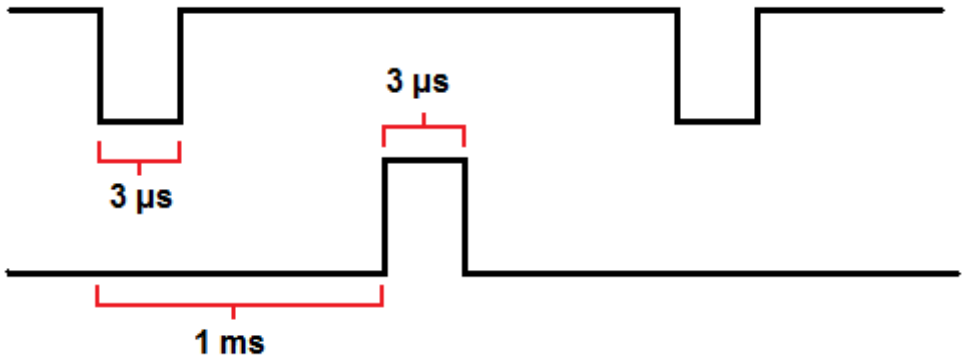


Figure 12. Flipping signals

The KMZ52 sensor typical sensitivity is 80 mV/(kA/m) at Vcc=5V. Considering a minimum earth field strength in the sensor plane of approximately 15 A/m, the sensor will deliver an amplitude of approximately 1.2 mV, when rotated in that field. The MAX1039 ADC chip has a voltage range for its analog input of ±Vref/2, where the external reference voltage is 2.048 V. As this is an 8 bit ADC, each step represents 2.048/256 V = 4mV. Therefore, some amplification is required in order to provide reasonable voltages to be read by the ADC chip. For example, the pre-amplifier might be designed to provide a gain of 103, thus allowing better utilize of the ADC's range (hence increasing the resolution in change in magnetic field strength). The communication between the MAX1039 ADC chip and the microcontroller is done by the Inter-Integrated Circuit (I²C) bus protocol. I²C uses only two bidirectional lines: Serial Data Line(SDA) and Serial Clock (SCL).

During the development of this thesis, an effort was made to interface the sensor to the ADC. The goal was to read the ADC's value using the I²C protocol. Despite a considerable

1 Note that since the start of this project NPX has removed the application note for KMZ51 and KMZ52

time being spent programming the MSP430F2618 microcontroller I was unable to get a response from the ADC, hence this task has been left for future work (see section 5.1). As a result of not being able to communicate with the ADC, the thesis re-focused on the software presented in the next section. The design of the circuitry (including the values of the capacitor, transceivers, resistor, and op-amps) is also left for future work.

4 Software applications

This chapter will present the description of an application that makes use of the information provided by a magnetic field sensor, as well as information about location. Different approaches of the same application will be examined and compared in the different sections, giving experimental results to support the most suitable solution. Concepts and technologies used to design the software are also described.

4.1 Orientation-aware map display

This application uses the information about the device's orientation provided by the KMZ52 sensor, that combined with the device's location coordinates, enables a PDA to display a map centered at the device's current position and rotated appropriately (depending upon the orientation of the device, i.e. showing in the superior part of the picture what should be in front of the user). The size of the portion of the map to be displayed (and the zoom factor of the image) can also be specified.

The initial version of the application relies on a web server to store the map and to generate the appropriate version of the map (as a picture) that is to be displayed. The web service is implemented in Hypertext Pre-processor (PHP) language. The web service receives the necessary parameters (x-y coordinates of the user, orientation angle, and image size) together with the URL that is requested by the user's client. We will assume that the web server already has stored the whole map in a file, thus it needs to generate a Portable Network Graphics (PNG) picture of the desired size, rotation, and location requested by the URL. The image is sent to the PDA's web client (i.e., the client's web browser or plug-in) to display it on the screen. The application running on the PDA sends a request and displays the image(s) received. In this initial version of the application most of the computational load is on the server. A prototype of the application was implemented for Microsoft's Windows Mobile using the .Net platform, and deployed on an HP iPAQ 5550.

The next section describes the solution in more detail, including how the map is stored at the server and how an image of a portion of the map at a specified location, orientation, and size is generated. However, before describing the processing we introduce the format used for storing the map.

4.1.1 Scalable Vector Graphics

Vector graphics uses geometrical primitives such as points, lines, curves, and shapes or polygons, which are all based on mathematical equations, to generate an image using computer graphics. Vector graphics formats are complementary to raster graphics, which encode images as an array of pixels. The main advantage of vector graphics over raster graphics, is that vector-based images can be scaled without degrading, while raster images

lose clarity when scaled up. If we want to achieve a reasonable clarity when enlarging a raster image, we need an original image that has very high resolution (hence takes a lot of storage), while vector images often take much less storage. Therefore, vector graphics are widely used in computer applications that use maps, since maps usually need to be scaled, vector graphics encode the map efficiently (minimizing memory requirements), and most applications do not need the detail that a raster map can provide. Some applications combine both vector graphics and raster graphics to enable rapid selection of a region of interest, followed by display of the relevant area in detail using a raster image (see for example, Google Earth).

The World Wide Web Consortium (W3C) standard for vector graphics is Scalable Vector Graphics (SVG). SVG images and their behaviours are defined in XML text files. This means that they can be searched, indexed, scripted and, if required, compressed. Since they are XML files, SVG images can be created and edited with any text editor. Additionally, specialized SVG-based drawing programs are also available for creating and manipulating SVG images.

4.1.2 Server SVG rendering

In our prototype the web server stores an SVG-format map of the third floor of the Electrum building in Kista, Stockholm, Sweden. Unfortunately, the Microsoft Compact Framework for mobile devices of .Net platform does not have built-in support for displaying SVG images, thus we need to render the SVG either locally ourselves or remotely (for example at the web server). In this approach we can render the SVG image as a PNG image (which can be displayed in the PDA) of the relevant portion of the map based upon the location and size parameters sent by the device. The center of the image will correspond to the x-y coordinates of the device's location. The resulting image should fit to the display, with its width determined by a size parameter. This rendering can be done using the vector graphics editor application Inkscape [23] or Batik [24]. Inkscape is distributed under a free software license, specifically GNU GPL. Inkscape has a command-line interface that enables it to be called from PHP code using the function *exec*. To implement the second approach above, we render the SVG image as a PNG image, by a command-line, such as:

inkscape.exe -f map.svg -e map.png -a x0:y0:x1:y1 -w1000 -h1000 -y 255

The original SVG file is specified after the option *-f* and the output PNG file is specified after the option *-e*. The parameters x0, y0, x1, and y1 specify the coordinates (in the original SVG image) of the rectangle that we want to render as PNG. The options *-w1000* and *-h1000* determine the width and height of the output PNG image in units of pixels (in the example the value is 1000, but can be set to any other value). Since the original image is a vector graphic, it is possible to take a small portion of the map and generate a scaled PNG image without any loss of quality (as would happen if we simply scaled a file in PNG format). The option *-y* sets the opacity of the background of the exported PNG (1 meaning full transparency, and 255 meaning full opacity). We need to set it to 255, because transparency caused some problems

when displaying the image on the PDA.

Batik is a Java-based toolkit for applications or applets that want to use SVG images. Specifically we will use the rasterizer utility. An example of a command-line using the Batik's rasterizer is:

java -jar batik-rasterizer.jar map.svg -w 300 -h 300 -a 400,400,200,200 -bg 255.255.255.255

This command takes the file *map.svg* and generates *map.png*. Options -*w* and -*h* specify the output width and height, and -*a* specifies the area of interest of the SVG file to rasterize (in the format *x*,*y*,*width*,*height*). The option -*bg* specifies the background fill color (in the format *alpha*.*red*.*green*.*blue)*. It is necessary to avoid alpha transparency for a proper visualization on the PDA screen.

The next step is to rotate the image. This process is described in the next section. It should be noted that the rotation can also be performed on the SVG data, then the rotated SVG image could be rendered as the final PNG file. Doing the rotation before rendering is much more efficient. This alternative means of producing the desired PNG image will be described in section 4.1.4.

4.1.2.1 Image rotation

PHP GD library [25], can be used to create and manipulate image files in a variety of different formats. With this library it is possible to read a PNG file, rotate the image by any desired angle, and save the resulting image as a new PNG file. The first thing that we have to take into consideration when rotating an image, is that if the angle of rotation is a multiple of 90º, then the rotation can be performed very easily. Otherwise the rotated image may include areas that are not in the original image, while other areas may be out of the clipping boundaries. Assuming that we want to rotate a picture that is part of a bigger map, we may need to generate a PNG image (as described in the previous section) that covers a larger area than specified by the desired size in order, to include the areas of the map that will be visible due to the rotation (see Figure 10). The question is how large this picture has to be to guarantee a successful rotation without having blank areas for which there actually is data.

The GD Library function *imagerotate* rotates the image in such a way that the rotated image preserves the original image (hence the image dimensions may be larger) (See figure 13).

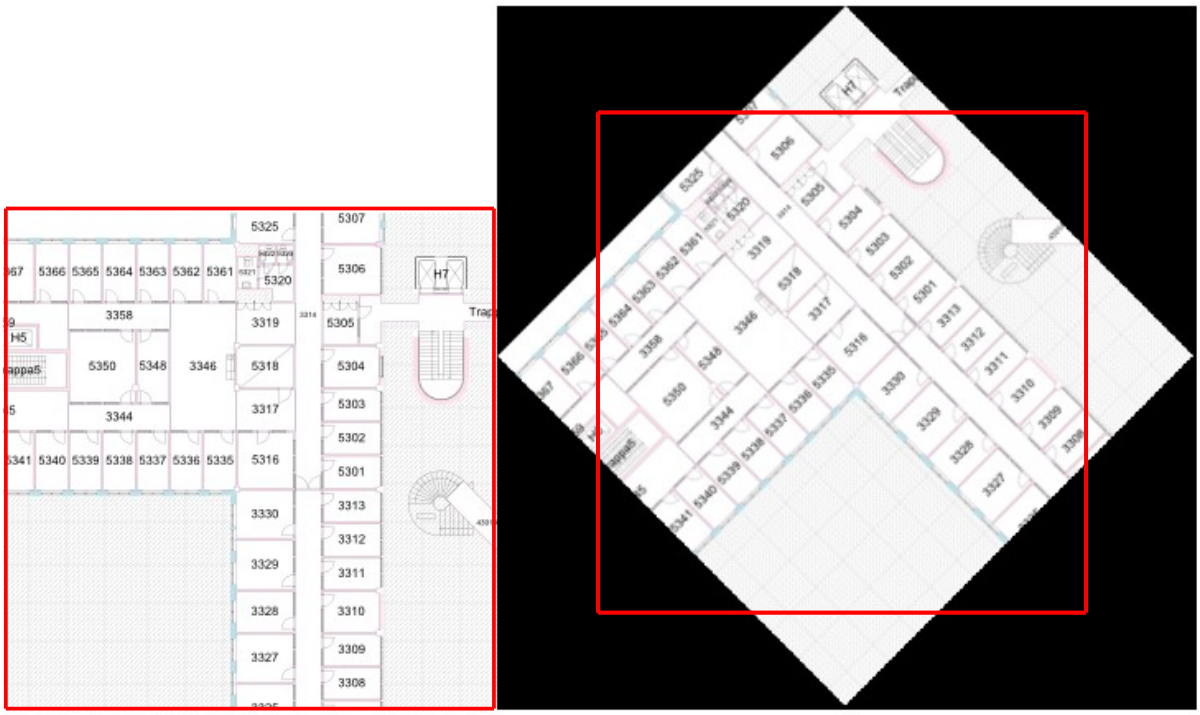


Figure 13. Rotation by GD Library. The first image is the original picture, and the second is the one generated by the GD function. The red rectangle represents the dimensions of the original image.

As we can see in the figure above, if we want to preserve size and zoom level without having empty areas, we need to use a larger original image, otherwise we will end up with the black areas shown in the figure. The new frame for the rotated picture resulting from the *imagerotate* function determines the minimum size needed for the original image. Figure 14 shows this with an example.

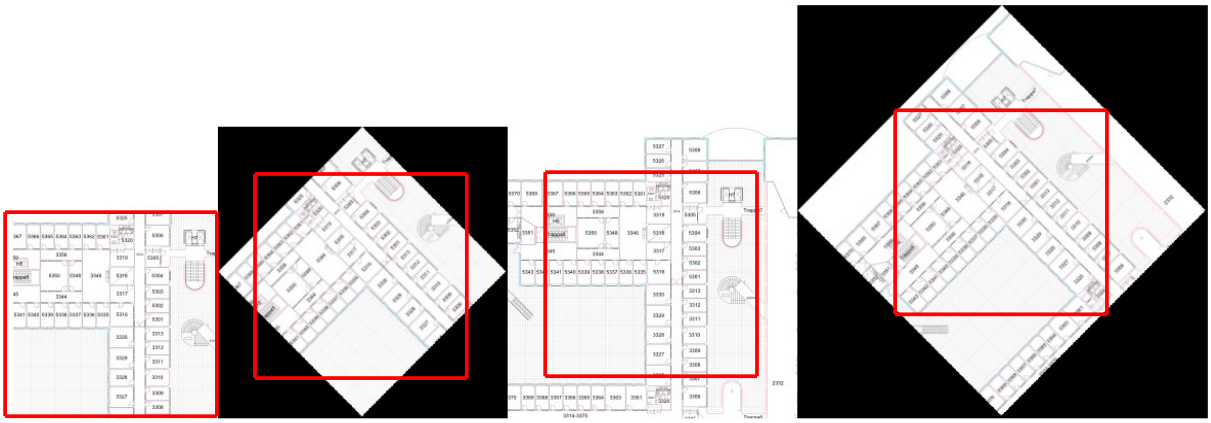


Figure 14. Example of rotation without empty space. (a) represents the desired size of the image (without rotation), (b) the rotated image containing the rotated image, (c) the large original image that will contain the desired rotated image, and (d) the rotated version of the desired image - before clipping out the desired subregion.

It is possible to calculate the minimum size required to avoid empty areas by using simple trigonometry. The rotated picture and the frame that contains it generate four right

rectangles, as it is shown in Figure 15. The angle α of the right rectangle is the angle of rotation of the picture. The length of one side of the frame is determined by the sum of the opposite leg (a) and adjacent leg of the triangle (b). Since we know the value of the hypotenuse (c, it is the size of the image to rotate) and the angle of rotation, we can calculate the minimum size required for a correct rotation (s), using trigonometric functions.

$$a = c \cdot \sin(\alpha); b = c \cdot \cos(\alpha);$$

$$s = a + b;$$

$$s = c \cdot \sin(\alpha) + c \cdot \cos(\alpha); s = c \cdot (\sin(\alpha) + \cos(\alpha))$$

The required size will be the biggest possible when the angle of rotation is 45º. In this case, the value of s will be $c \cdot ((\sin(45º) + \cos(45º))$ *(i.e.,* $1.41 \cdot c$*)*. A picture that covers an area 1.41 times bigger than the original one will be sufficient to allow a rotation in any angle and still extract a square picture of the same size as the non-rotated one.

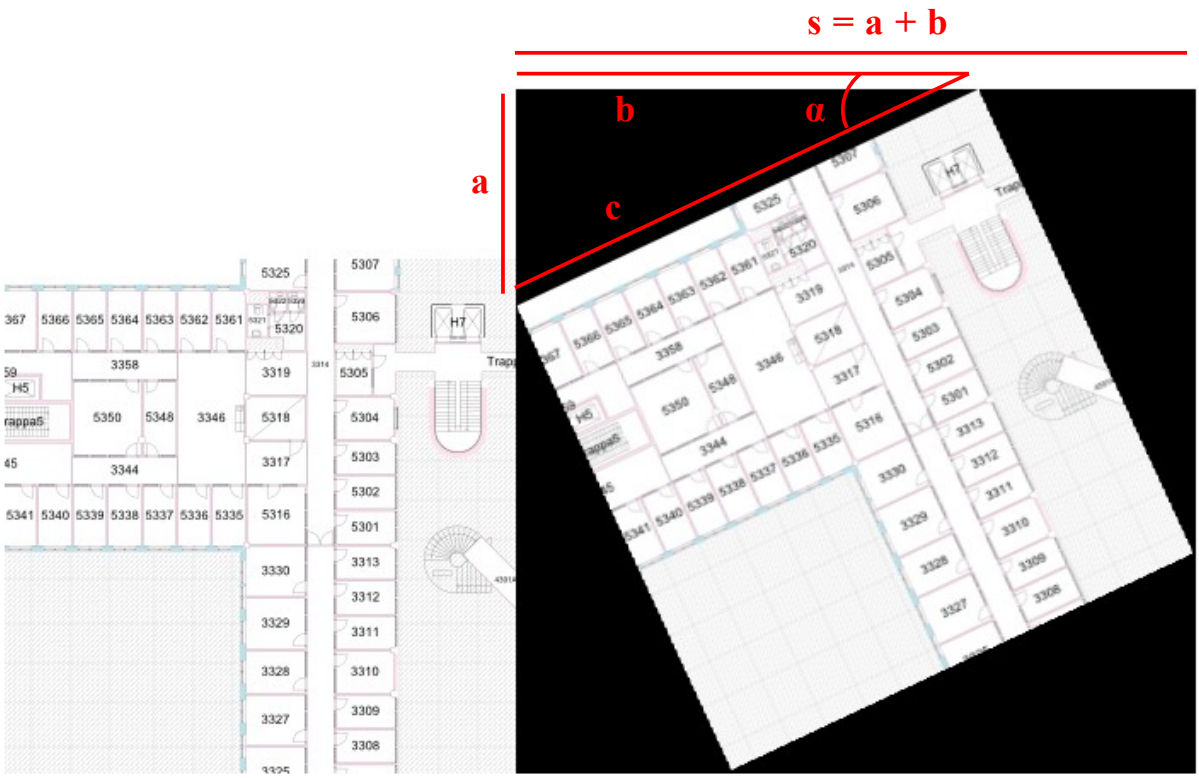


Figure 15. Trigonometric representation

Once the rotated picture has been generated, we can cut a square portion in the image with the same size of the non-rotated picture, using the GD Library function *imagecopyresampled*. After this, the picture is ready to be sent and displayed on the screen of a PDA.

4.1.2.2 Process summary

The process (involving the PDA application and the web service) to display the map with the proper rotation can be summarized in the following steps:

1. The PDA sends a web the request to the server, specifying in the URL of the request the value of the parameters of **rotation angle**, **x-y position** of the center of the image, and **size** of the area to display. An example of such a request is:

   http://192.168.2.2/mapdisplay.php?a=45&x=600&y=600&tam=200

2. The web service extracts the parameters and executes the following steps:
   - Call the *Inkscape* or *Batik* editor using the command-line interface. The goal is to obtain a PNG format image of a piece of the map determined by the values of position and size. The size of the image converted to PNG is 1.5 times bigger than the size requested, since this value assures rotation without empty spaces (in the worst case must be 1.41 times bigger). See Figure 16 (b).
   - Rotate the PNG image using the GD library for PHP. See Figure 16 (c).
   - Cut the image rotated to get an image of the size requested. The center of this image must be the same as the center of the image before being cut. See Figure 16 (d).
   - Send the image with the requested size in the HTTP response.
3. The PDA application receives the picture's stream as a response, and displays the image in a picture box.

Source code for using both Inkscape and Batik are shown in Appendix B.

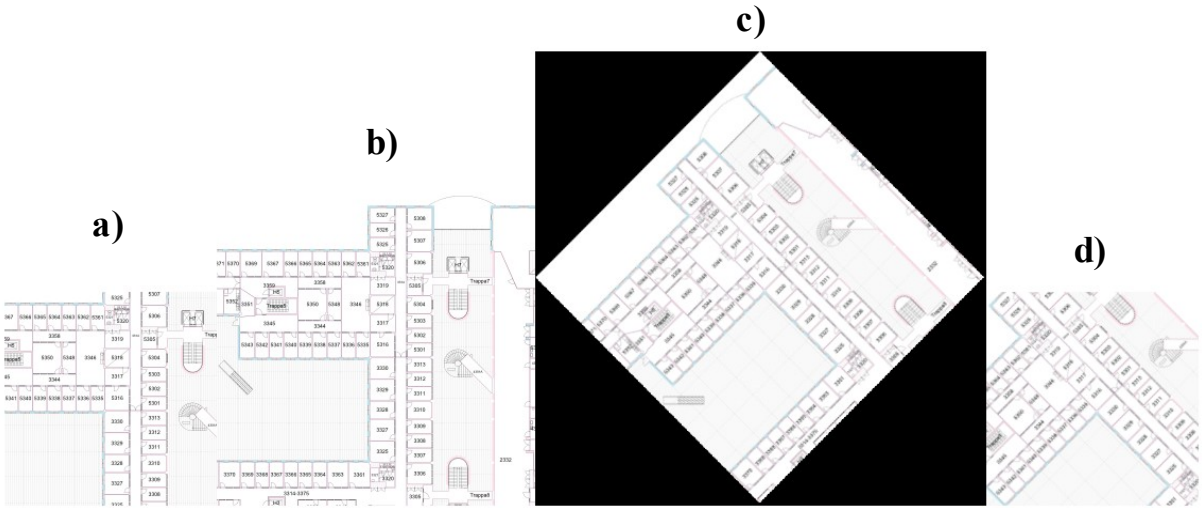


Figure 16. Example of execution (change from 0º to 45º).
a) Image displayed previously on a PDA screen (angle 0º)
b) Image generated by Inskape editor, covers an area 1.5 bigger than the image displayed
c) Image rotated 45º
d) Image cut to the original size (the center point keeps being the same)

4.1.2.3 Image rotation on the SVG data

The rotation process using PHP and the GD library that was described in section 4.1.2 can be easily performed by making changes in the SVG file that represents the image, before rendering it as a PNG file. This approach avoids worrying about the blank areas that could appear when rotating a square picture. The option "*g transform*" in the XML file allows us to perform several transformations on the entire image if the option encapsulates all the definitions in the XML file. Specifically we will use the transformations of rotation and translation. An example of the use of this option is:

```
<g transform="translate(600,400), rotate(90)">
        xml image definition
</g>
```

The axis of rotation used by *rotate* is placed at the centerpoint (0,0) using a *translate*, and the rotation is done clockwise. We need to rotate the image around an specific point, that will be determined by the current location, hence before performing the rotation we need to translate the image to assure that the current location point is placed at the centerpoint. After the rotation the inverse translation can be done, and the current position point will have the same value as before in the SVG file. Note that the rest of the points in the image (out of the axis of rotation) may change because of the rotation. The order of the transformations that are made with "*g transform*" is from right to left. This example shows how a rotation of 45º can be made if the current location is placed at the coordinates (300,200):

```
<g transform="translate(300,200), rotate(45), translate(-300,-200)”>
        xml image definition
</g>
```

Changes to the SVG files can be easily done by PHP using filesystem functions such as *fgets* and *fwrite*.

After this we only need to render the SVG file with the desired size, assuming that the center of the rendered image must be the chosen location.

The application that runs on the PDA can be used with both solutions (rotation by PHP or SVG), as in both cases the server generates a PNG image. The web service's source code for rotation in SVG is simpler than PHP rotation, and this code is included in Appendix B.

Table 5 compares both Inkscape and Batik rasterizer tools. The table shows the time spent by the server to generate the picture to be sent and displayed on the PDA, depending on the size of the frame to be displayed. Inkscape provides faster execution when the size of the frame is smaller than 400. For larger frames Batik is a faster solution. However, Batik offers lower image quality than Inkscape (see Figure 17). The measurements were made over 10 executions in a laptop machine (4 GB of RAM) with a Windows Vista OS running on a CPU Intel Core 2 Duo T6600 running at 2.2 GHz. The SVG file used in the measurements represents a map of the floor 3rd of Electrum building. Its size is 1840 KB and has a number of different SVG elements. It has 9129 lines of data, and 1927871 characters. Simpler SVG files will have considerably faster rendering times.

Table 5. Timing comparison between Batik and Inkscape for rendering a sample map

Size (pixels)	**Inkscape (seconds)**			**Batik (seconds)**		
	Average	Minimum	Maximum	**Average**	Minimum	Maximum
50	**5.027**	4.681	5.512	**7.059**	6.889	7.289
100	**5.067**	4.930	5.130	**7.078**	6.845	7.336
200	**5.975**	5.653	6.475	**7.050**	6.939	7.283
400	**7.126**	6.978	7.361	**8.103**	7.748	8.446
800	**8.925**	8.661	9.309	**8.107**	7.782	8.780

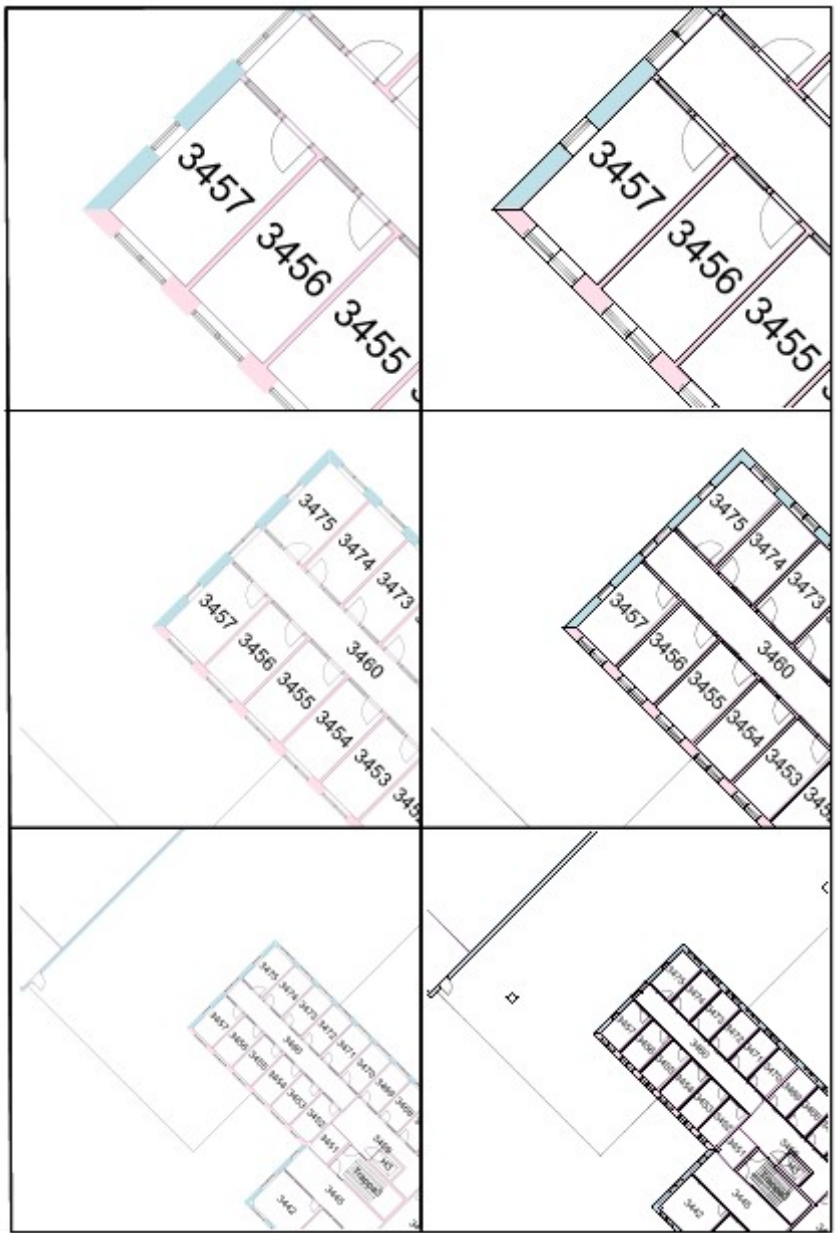


Figure 17. Image quality comparison between Batik and Inkscape. Sizes of 50, 100 and 200 pixels. Angle of 45º. Inkscape: left, Batik: right.

Table 6 compares the solution that performs the rotation by modifying the SVG file and the solution that rotates the PNG rasterized image. In the experiments, the rasterization of the SVG file has been made in both cases by Inkscape. The time obtained are similar as much of the computation is rendering the image, while in comparison the rotation of a PNG image is fast. The measurements were made over 10 executions in a laptop machine (4 GB of RAM) with a Windows Vista OS running on a CPU Intel Core 2 Duo T6600 with 2.2 GHz of speed. The SVG file used in the measurements represents a map of the floor 3rd of Electrum building. Its size is 1840 KB and has a number of different SVG elements. It has 9129 lines of data, and 1927871 characters. Simpler SVG files will have considerably faster rendering times.

Table 6. Timing comparison between SVG rotation and PNG rotation

Size (pixels)	SVG rotation (seconds)			PNG rotation (seconds)		
	Average	Minimum	Maximum	**Average**	Minimum	Maximum
50	**5.027**	4.681	5.512	**5.175**	4.751	5.786
100	**5.067**	4.930	5.130	**5.131**	5.043	5.245
200	**5.975**	5.653	6.475	**6.178**	5.897	6.345
400	**7.126**	6.978	7.361	**7.393**	7.023	7.656
800	**8.925**	8.661	9.309	**9.602**	9.334	9.909

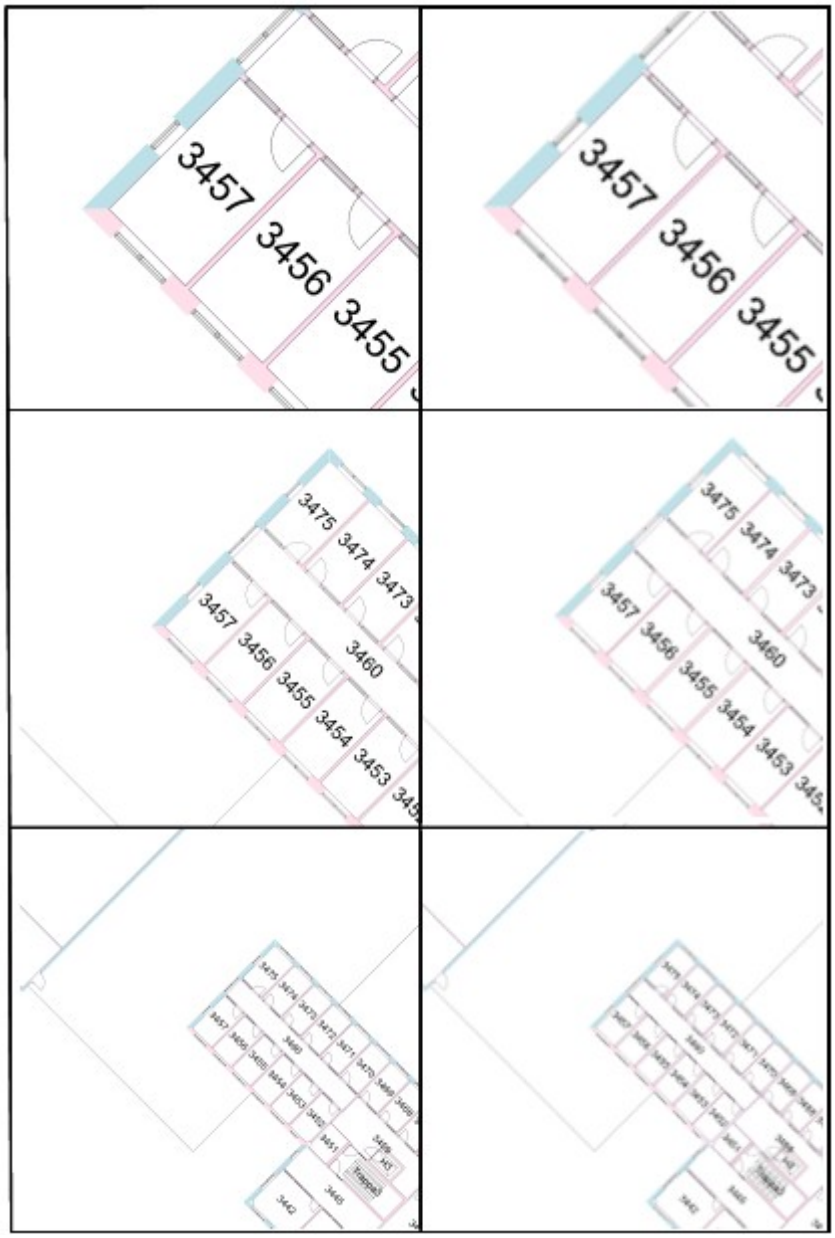


Figure 18. Image quality comparison between SVG rotation and PNG rotation. Sizes of 50, 100 and 200 pixels. Angle of 45º. SVG rotation: left, PNG rotation: right.

The rotation of a PNG image causes distortion and a loss of quality in the final output

(see Figure 18).

With any of the approaches described above based on a web server performing all the computation, it is necessary to rasterize the SVG image every time a change in orientation, size or position is requested. This makes the process very slow. Delays of 5, 6, or 7 seconds are not acceptable in an application that sends requests very frequently as such an application is likely to be used while on the move, with variations of orientation and position occuring in short periods of time. In fact, rotating the device about its center (which should only require a rotation) takes as long as rendering a new image for a new location and orientation. This unacceptable performance motivated us to think of using a client-based solution in which the extraction of a SVG image would be performed once, and all later changes would be computed on the client side. This means that the client has to store the vector information and render it on the display. This solution will be examined in section 4.1.5.

4.1.3 Client SVG rendering

4.1.3.1 SVG for mobile devices

Small devices are a target area for vector graphics display. In order to meet these demands the SVG Working Group made an effort to create a profile specification that addresses mobile devices. To address the range of different device families, two profiles were defined. The first low-level profile, SVG Tiny is suitable for highly restricted mobile devices such as cellphones. The second profile, SVG Basic is targeted for higher level mobile devices such as PDAs. Mobile SVG introduces constraints on content, attribute types, properties, and user agent behavior, because of the low memory, low CPU power and limited display of mobile devices.

The most successful implementations for cellphones are developed by Ikivo [26] and Bitflash [27], while for PDAs, Bitflash and Intesis [28] have popular implementations. Adobe Flash Lite has optionally supported SVG Tiny since version 1.1.

Some mobile SVG players such as the ones from Ikivo and BitFlash come pre-installed, the manufacturers burn the SVG player code into their mobiles before shipping to the customers. There are also some web browsers for mobile devices (such as Opera Mini and the iPhone's Safari) that include SVG support.

The level of SVG Tiny support available varies from mobile to mobile, depending on the SVG engine installed. Many newer mobile products support additional features beyond SVG Tiny 1.1, like gradient and opacity. The iPhone, for example, supports declarative animation but not interactivity.

4.1.3.2 Implementation of a SVG rasterizer

The idea of implementing our own rasterizer of SVG images involves reading the XML file that represents the map, storing the vector information, and drawing on the screen the objects defined by the information previously stored.

The SVG specification includes a set of features which can be exploited in order to gain speed when displaying the picture, since the application is designed to show maps, which do not need the level of detail that is provided by all of the objects. In order to avoid the client having to deal with information that will not be necessary when rendering the map on the display of the PDA (reducing the amount of information reduces the time needed for parsing, the amount of time and space needed for processing the map), a web server can filter the SVG file, sending to the PDA only an SVG image with relevant information. The filtering process on the web server can exclude objects whose small size would not be displayed on the PDA (hence they have little relevance). The client can specify the the level of detail and precision when requesting the map. The level of detail can be specified by setting a limit on the size of a bounding box for the object, thus the server can exclude objects smaller than this value. Once the SVG file has been parsed by the client and the information stored, the client does not need to receive or process a new SVG file unless a change in the precision is requested in a new request to the server, since variations of position, scale, and orientation can be performed using the vector information stored locally. Note that a large change in the location of the device, will necessitate reading a new set of data - concerning the SVG representation for objects around the new location.

Filtering the SVG picture to produce a simpler representation is a straight-forward idea, but implementing it requires further knowledge of the SVG standard. The first step is to separate the objects that are more representative in the image of the ones that can be saved with a minor loss of quality. Normally in a map (either a street or a building map) the most important elements are formed by straight lines. The following features of the SVG standard can be left out of our limited implementation:

- **Painting**: SVG shapes can be filled and/or outlined (painted with a color, a gradient, or a pattern). Fills can be opaque or have various degrees of transparency.
- **Gradients and Patterns**: SVG shapes can be filled or outlined with solid colors as above, or with color gradients or with repeating patterns.
- **Clipping, Masking, and Compositing**: Graphic elements, including text, paths, basic shapes, and combinations of these, can be used as outlines to define both inside and outside regions that can be painted independently.
- **Text**: Unicode character text included in an SVG file is expressed as XML character data. Many visual effects are possible. All text will be left out in our implementation.

Specifically we have focused on the SVG element called ***path***. Paths describe a series of connected points, and how connections between those points are drawn, be they straight lines

or a variety of curved ones. The syntax used to describe a path is explained in the following paragraphs.

Quoting the specification, "A path is defined by including a 'path' element which contains a d="(path data)" attribute, where the *d* attribute contains the moveto, line, curve (both cubic and quadratic Béziers), arc, and closepath instructions." The path is defined in the 'd' attribute of the 'path' tag by a string of white space seperated commands and coordinates. Path commands are case-sensitive. An uppercase command's points use absolute postioning and a lowercase command's points are relative to the last point. The one exception to this is the first point always uses absolute positioning.

A path is basically formed by *moveto* and *lineto* point commands. If we think of a pen drawing on a sheet of paper, a new moveto point places the pen at the given coordinates without drawing any line. A lineto point draws a line between the current position of the pen (determined by the last command: either a moveto or a lineto command) and the new coordinates given (which will become the current position). There are also commands for drawing quadatric and cubic Bézier curves, and elliptical arc curves, but in our limited implementation for the map display these commands are not implemented.

The following commands (from the W3C SVG specification [29]) are implemented in our implementation:

Table 7. SVG Path commands

Command	Name	Description
M (absolut) **m** (relative)	Moveto	Start a new sub-path at the given (x,y) coordinate. **M** (uppercase) indicates that absolute coordinates will follow; **m** (lowercase) indicates that relative coordinates will follow. If a relative moveto (**m**) appears as the first element of the path, then the point is treated as a pair of absolute coordinates. If a moveto is followed by multiple pairs of coordinates, the subsequent pairs are treated as implicit lineto commands.
L (absolute) **l** (relative)	Lineto	Draw a line from the current point to the given (x,y) coordinate which becomes the new current point. **L** (uppercase) indicates that absolute coordinates will follow; **l** (lowercase) indicates that relative coordinates will follow. A number of coordinates pairs may be specified to draw a polyline. At the end of the command, the new current point is set to the final coordinates.
H (absolute) **h** (relative)	Horizontal lineto	Draws a horizontal line from the current point (cpx, cpy) to (x, cpy). **H** (uppercase) indicates that absolute coordinates will follow; **h** (lowercase) indicates that relative coordinates will follow. Multiple x values can be provided (although usually this does not make sense). At the end of the command, the new current point becomes (x, cpy) for the final value of x.

V (absolute) **v** (relative)	Vertical lineto	Draws a vertical line from the current point (cpx, cpy) to (cpx, y). **V** (uppercase) indicates that absolute coordinates will follow; **v** (lowercase) indicates that relative coordinates will follow. Multiple y values can be provided (although usually this does not make sense). At the end of the command, the new current point becomes (cpx, y) for the final value of y.
Z or **z**	Closepath	Close the current subpath by drawing a straight line from the current point to current subpath's initial point.

4.1.3.2.1 Web server's filtering

The web server application that has been implemented takes any SVG file containing a map and generates a new file containing only straight lines defined by the commands in table 7 in a path element. It also removes all lines that are shorter than a bounding box determined by a given length. This minimum value is specified by the client when it sends the parameter ("minLength") in the URL of the web request. This length is given in SVG user units. According with the SVG specification, a "user unit in the initial coordinate system is equivalenced to the parent environment's notion of a pixel unit" (See [30]). This filtering process reduces the size of the SVG file to be read by the client, while limiting the loss of clarity of the picture that will be displayed. Additionally, the web service also transforms all the relative coordinates (lower-case commands) to absolute coordinates, in order to simplify the parsing process in the client (as we need to store the information about lines in absolute coordinates, the conversion from relative to absolute can be done in the server).

Table 8 shows the reduction in size for a SVG file that represents the 3rd floor of Electrum Building. The original size of the file is 1840 KB (Note that this is the same file that was used earlier for the rasterizer performance measurements)

Table 8. Sizes after filtering an SVG file

Minimum length (user units)	**Size of the file (KB)**	**Minimum length (user units)**	**Size of the file (KB)**
Original file	1840	10	168
0	1210	11	159
1	639	12	152
2	552	13	139
3	475	14	133
4	405	15	125
5	332	16	119
6	223	17	112
7	208	18	106
8	190	19	103

9	178	20	95

Table 9 shows the savings in size of the output file achieved with the filtering process for different minimum lengths. Savings are presented in relation to the original file, and in relation to the previous output file. For the specific SVG file used in the experiments, we can reach some conclusions if we analyze the statistics. The largest savings are achieved when filtering the smallest lengths, as this particular map has an important level of detail and contains many tiny objects. Filtering lengths larger than 6 user units does not produce a big difference in size in relation with previous minimum lengths.

Table 9. Filtering statistics

Minimum length (user units)	**Saving relative to previous**	**Cumulative saving**	**Minimum length (user units)**	**Saving relative to previous**	**Cumulative saving**
Original file			10	0.54%	90.87%
0	34.24%	34.24%	11	0.49%	91.36%
1	31.03%	65.27%	12	0.38%	91.74%
2	4.73%	70.00%	13	0.71%	92.45%
3	4.18%	74.18%	14	0.33%	92.77%
4	3.80%	77.99%	15	0.43%	93.21%
5	3.97%	81.96%	16	0.33%	93.53%
6	5.92%	87.88%	17	0.38%	93.91%
7	0.82%	88.70%	18	0.33%	94.24%
8	0.98%	89.67%	19	0.16%	94.40%
9	0.65%	90.33%	20	0.43%	94.84%

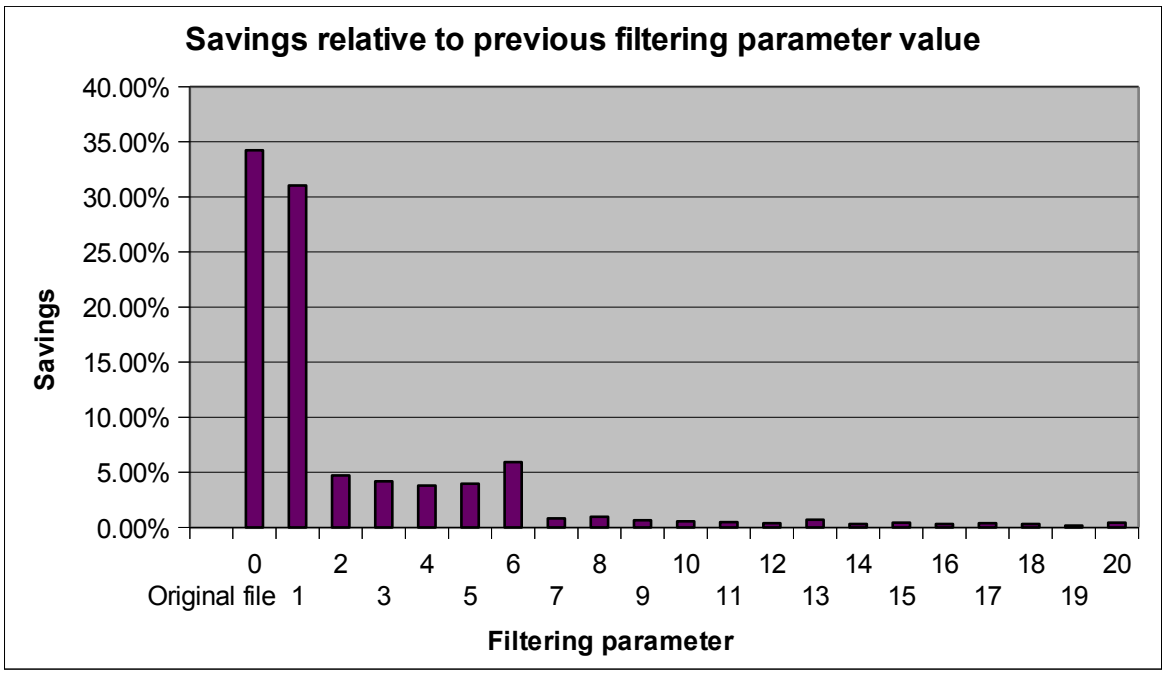


Figure 19. Savings relative to previous parameter value

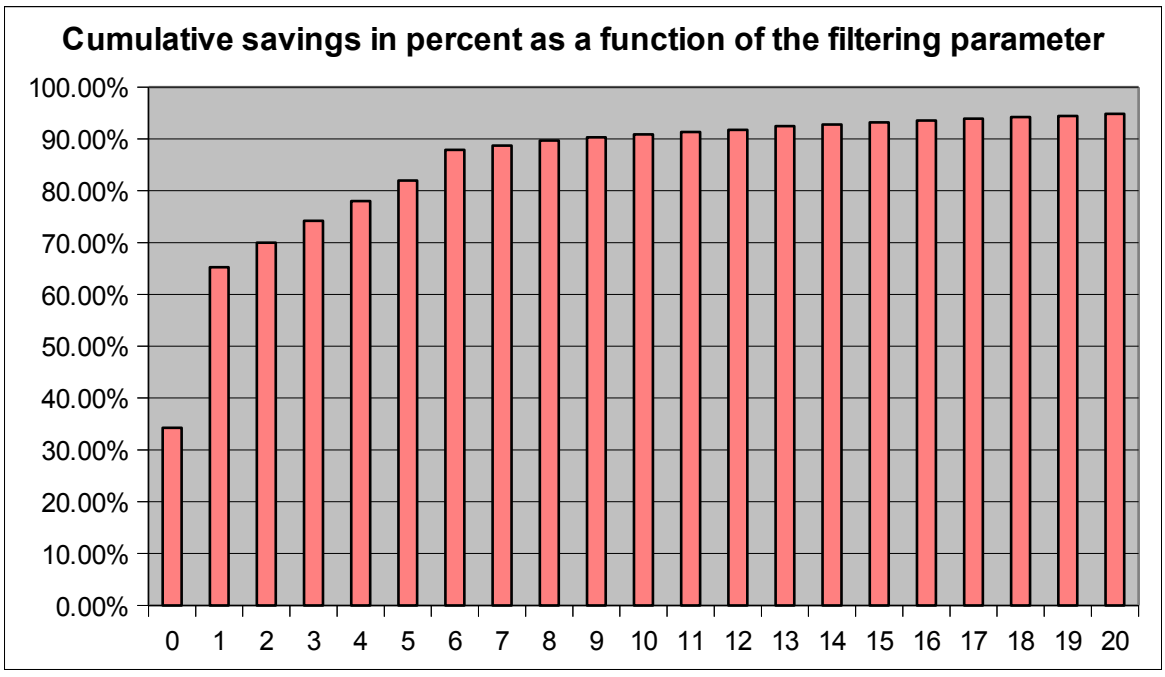


Figure 20. Cumulative savings

The Figure 21 shows a comparison between a map (floor 3rd of Electrum building) filtered by the server application with different filtering values, as it is displayed on a web browser. (a) is the original SVG map. (b), (c), (d), (e) and (f) are the map filtered with minimum length values of 1, 5, 10, 15, and 20 user units respectively. As it can be seen, for this particular map, values larger than 15 remove **relevant** lines of the picture. An appropriate

compromise between image quality and efficiency has to be achieved for every picture. As the image quality is a factor highly dependent on the user's subjectivity, there is not a general way to achieve an optimal compromise, but at least we can determine when an increase of the minimum length parameter is not relevant in efficiency, as we can see in the previous table and graphics.

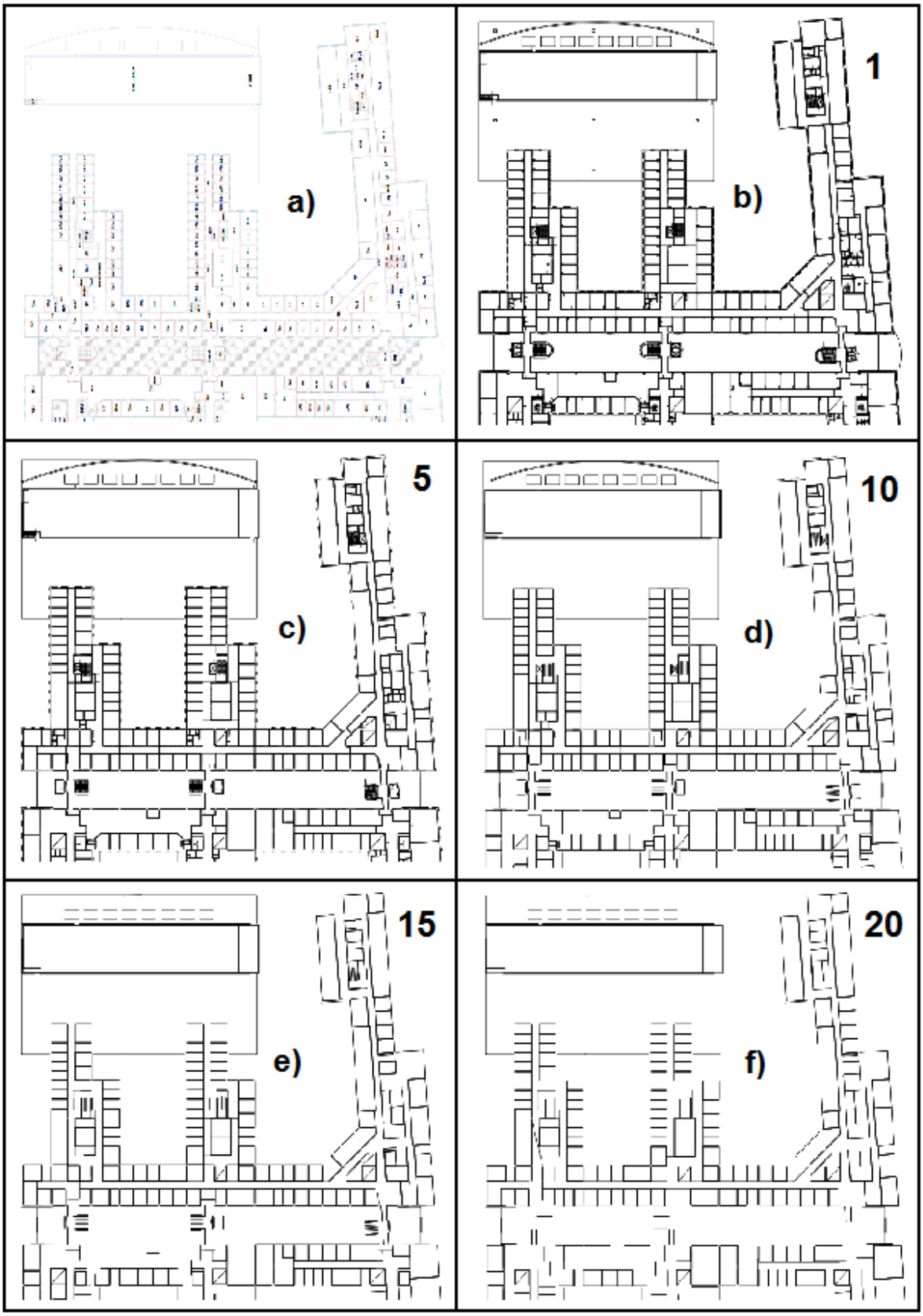


Figure 21. Appearance of a map after filtering with different filtering parameters

Table 10 shows the time the web server takes to generate a new SVG file from the

original 1840 KB file (the same as in the previous measurements: floor 3rd of Electrum building). The measurements were made over 10 executions in a laptop machine (4 GB of RAM) with a Windows Vista OS running on a CPU Intel Core 2 Duo T6600 with 2.2 GHz of speed. The processing time decreases slightly when a larger filtering parameter is demanded, as the larger filter parameter leads to fewer paths in the output file. Once a new reduced map is generated, if the same filtering parameter is requested for the same map, the file can be cached and the cached version served to the client **without** any computation. Simpler SVG files will have considerably faster filtering times.

Table 10. Execution times of the web server application

Minumum length (user units)	**Time (seconds)**		
	Average	**Minumum**	**Maximum**
1	**3.043**	2.941	3.323
5	**3.033**	2.921	3.294
10	**3.007**	2.834	3.298
15	**2.980**	2.872	3.138
20	**2.853**	2.790	3.031
25	**2.800**	2.734	3.003

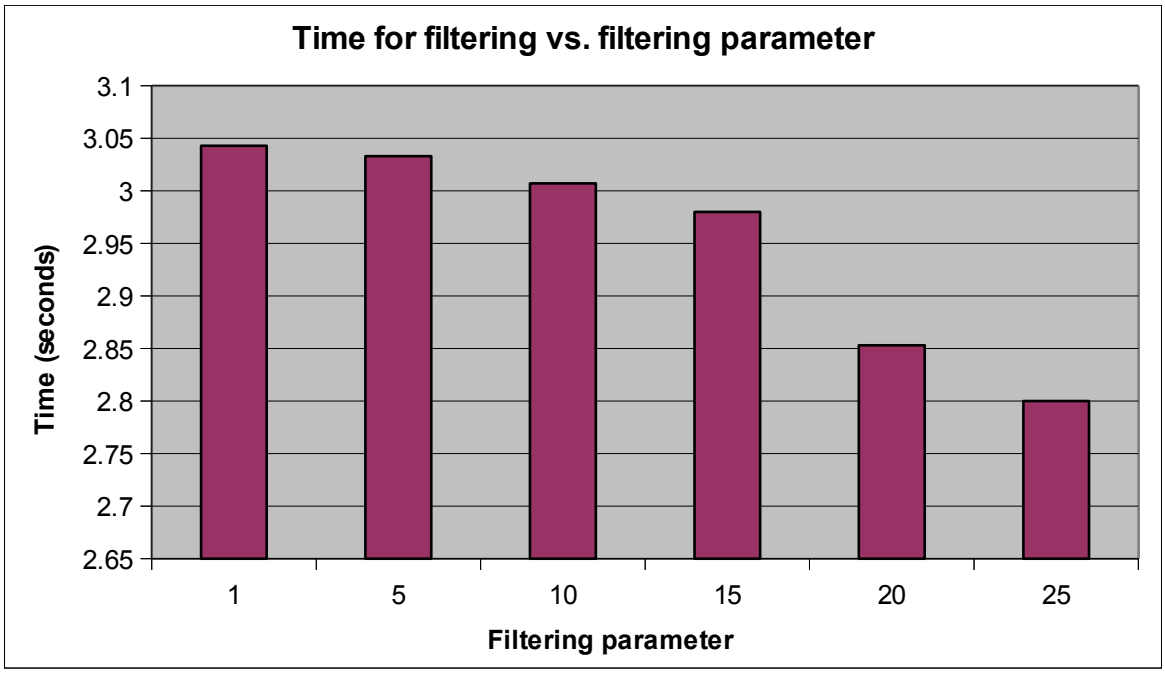


Figure 22. Time for filtering vs. filtering parameter

4.1.3.2.2 Client application

When the client receives the filtered picture, it is a XML-based format document and has to be parsed. The path syntax has to be interpreted and the information corresponding to

the lines to be drawn has to be stored in a data structure. This process is quite expensive with regard to execution time, but once the information is stored any portion of the map can be displayed in any scale, angle, or position.

As it has been explained in previous paragraphs, the *path* syntax give us a series of coordinates which some of them are connected to form lines. The connections are determined by the *moveto, lineto* and *closepath* commands. The implementation stores each separate line in its own structure. Each line is represented by an array of points (x and y coordinates). The line starts with the first point in the array and connects each point in the order of increasing index in the array. The whole picture is stored as an array of lines. This structure is shown in Figure 17. If the XML parser finds a *path* element with a *d* attribute "*M 10 10 H 20 V 20 L 30 30 Z M 50 50 L 0 0 L 10 20",* two new arrays of points "(10,10) (20,10) (20,20) (30,30) (10,10)" and "(50,50) (0,0) (10,20)" will be generated and included in the array of lines.

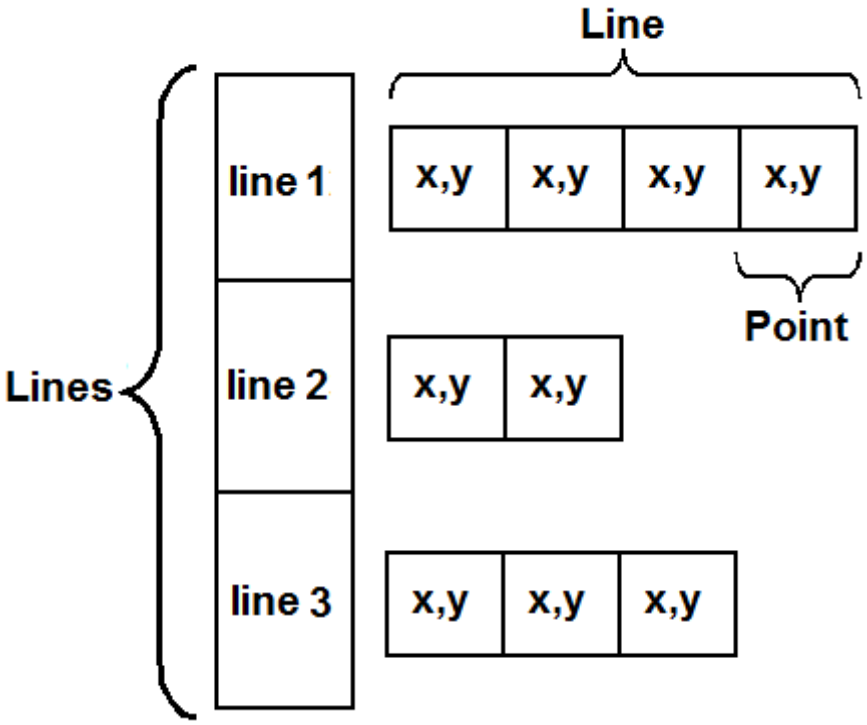


Figure 23. Structure used to store the data of the picture

The task of drawing the lines can be done by iterating along the arrays of lines. The *Graphics* class of .Net implements a function to draw a line between two given two-dimensional points. As the client application is running on a PDA, changes of orientation and position could happen very frequently. The user also may change the zoom level (i.e. the scale of the map) at any time. The goal of the client application is to provide the fastest possible display of the map. Thus, it is important to calculate and display only the lines that are going to be visible when the map is shown, given the current scale factor, the angle, and the position values.

The computation that must be applied to every point consists of a translation equivalent of placing the current location in the center of the frame, and a scaling and rotation in relation to the current location. To perform the scaling and rotation the current point must be placed in the (0,0) coordinates, hence the order for the transformations is: translation equivalent to place current point in (0,0), scaling, rotation, and translation equivalent of placing (0,0) in the center of the frame.

Figure 24 shows these transformations. The green spot represents the current location point, and the blue spot is the point whose new position we want to calculate. The frame in which the map is displayed is represented by the red square, whose left-superior corner is the (0,0) point. Figure 24 (a) is the picture as it is stored in the PDA after parsing the SVG file. In the current configuration of the application, the current location is placed on the green spot, and a reduction in scale and a rotation of 90º have been applied. We want to calculate the position for the blue spot point from the map information stored in the PDA (original coordinates). Figure 24 (b) is the step in which the blue point is translated with the same displacement that makes the current location to be placed in (0,0). In Figure 24 (c) the rotation and scaling is performed on the point, and in Figure 24 (d) the blue point is translated to its final position.

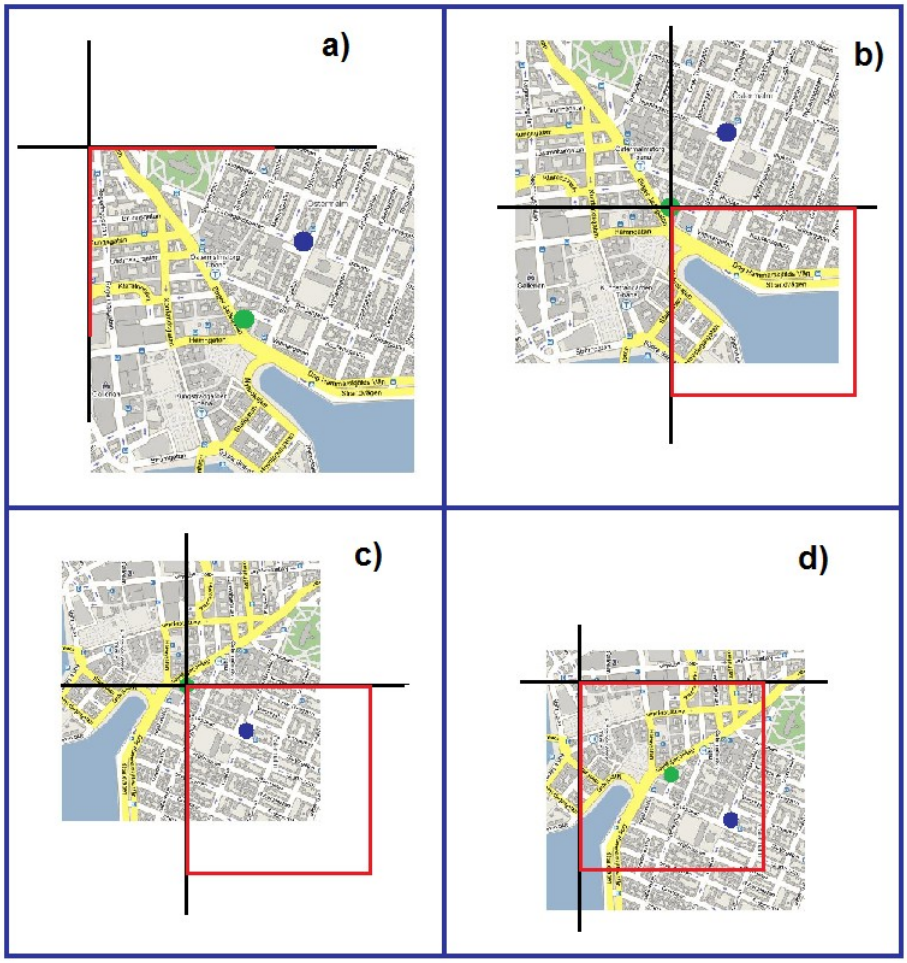


Figure 24. Transformations for a specific point

In order to display the picture more rapidly, it is possible to calculate only the position of the points that are going to be (or are very likely to be) inside the frame, by computing a clipping of an area of interest. Depending on the values of scale, rotation, and position, a clipping frame can be defined that will determine which points have to be calculated and displayed.. The clipping frame can be estimated by performing the inverse scaling, rotation, and translation to the displaying frame, calculating the coordinates of the four corners of the frame. To estimate if a given point is inside the clipping frame, we can discard the points that will be out of the box containing it (see Figure 15 in section 4.1.2).

As it is shown in Figure 25, the box containing the clipping frame is defined by two points (x_1,y_1) and (x_2,y_2). The following equations define these values, where x and y are the coordinates of the current location. The origin of this formula has been explained in section 4.1.2:

$$x_1 = x - (\text{size}/2) * (1/scale) * (\sin(angle\,\%\,90) + \cos(angle\,\%\,90))$$

$$y_1 = y - (\text{size}/2) * (1/scale) * (\sin(angle\,\%\,90) + \cos(angle\,\%\,90))$$

$$x_2 = x + (\text{size}/2) * (1/scale) * (\sin(angle\,\%\,90) + \cos(angle\,\%\,90))$$

$$y_2 = y + (\text{size}/2) * (1/scale) * (\sin(angle\,\%\,90) + \cos(angle\,\%\,90))$$

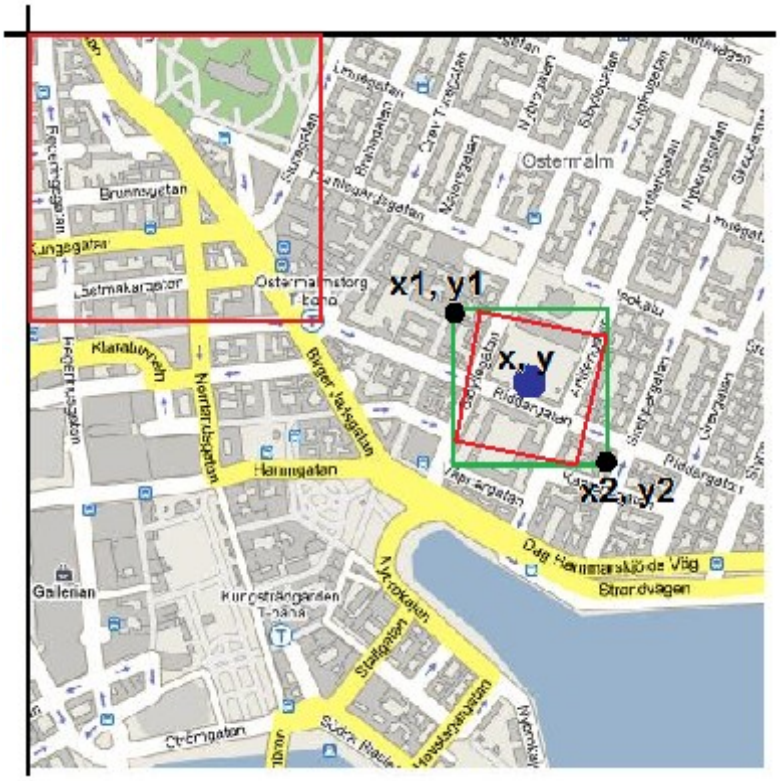


Figure 25. Box containing the clipping frame (scale=2, angle=20º)

With the current position in the center, the size of the box containing the clipping frame is determined by the inverse of the scale (as it is bigger a smaller portion of the map is visible on the screen) and the rotation angle. Depending on the angle the size of the box increases, with its maximum when the angle is multiple of 45º. If a point is inside this box, more

accurate operations have to be performed to check if the point is inside the rotated clipping frame, but we can save this computation is the point is out of the frame determined by (x_1,y_1) and (x_2,y_2). If any of the end points of a line are inside the clipping frame, the line will be displayed. To check if a point is inside the rotated clipping frame, we can calculate its final position after the transformations, and only display it if is inside the displaying frame. As it will be explained further (see Table 13), not drawing a line saves an important execution time.

This approach makes the display of a map generally fast as only the lines that are going to appear in the screen are calculated and displayed. The process will be faster as the zoom level increases because less lines have to be drawn on the screen. The time that it takes to display a picture depends on the number of lines that are calculated and displayed, and this number varies according to the zoom level, orientation angle, position, characteristic of the map, and minimum length that is requested to the web server.

The following table shows the times that it takes to display a map (the same as in the previous measurements: floor 3rd of Electrum building) that is already stored in a HP iPAQ 5550, for different zoom levels and different filtering values in the SVG file generated by the server. Lines shorter than the minimum length value are removed. The table also shows the number of lines displayed on the screen in each case. Tests have been done for a location (350, 350) and an angle of 45º. The times obtained depend directly on the number of lines calculated and displayed, but different current locations can make the times change considerably, as some areas of a map can contain much more lines than other areas. However, for a given current point of the same map, an increase of the scale will reduce (or at the most it will stay the same) the number of lines to display, because some of them will be out of the clipping frame. An increase in the filtering parameter for the same location point and scale factor will always drive to a reduction of the lines to calculate and, consequently, of the displaying time.

In this case, for scale values larger than 2 we get reasonable times for this kind of application. For smaller values, it depends on the filtering parameter if the response time is acceptable. A minimum length value of 10 user units is regard to be appropriate (see Figure 21) and gives reasonable response times for scale values 1.5 and 1.25 (Figure 26 shows how the sample map is displayed for location (350,350), an angle of 45º, and a scale value of 2). The application is likely to run generally with a scale value larger than 1, and not to show the entire map, hence we think the results obtained are quite acceptable.

Table 11. Display times on the client

	Min. Lenght = 5		Min. Lenght = 10		Min. Lenght = 15		Min. Lenght = 20	
Scale	**Time (ms)**	**Lines**	**Time (ms)**	**Lines**	**Time (ms)**	**Lines**	**Time (ms)**	**Lines**
1	2190	3650	1160	1899	823	1290	635	987
1.25	1413	2449	768	1312	505	881	404	698
1.5	914	1489	537	842	398	557	296	477
1.75	594	898	369	563	264	351	212	316
2	477	636	313	428	187	254	174	233
2.25	391	436	243	299	151	178	138	164
2.5	332	360	210	245	135	151	127	150
2.75	292	313	187	215	130	137	118	136
3	256	262	161	176	114	116	104	116
3.25	232	216	143	144	98	91	90	91
3.5	216	188	133	125	92	78	83	78
3.75	192	159	125	109	85	69	76	69
4	184	139	109	93	76	57	71	57
4.25	174	116	103	81	72	49	62	49
4.5	169	106	96	71	68	41	60	41
4.75	156	85	94	59	63	33	53	33

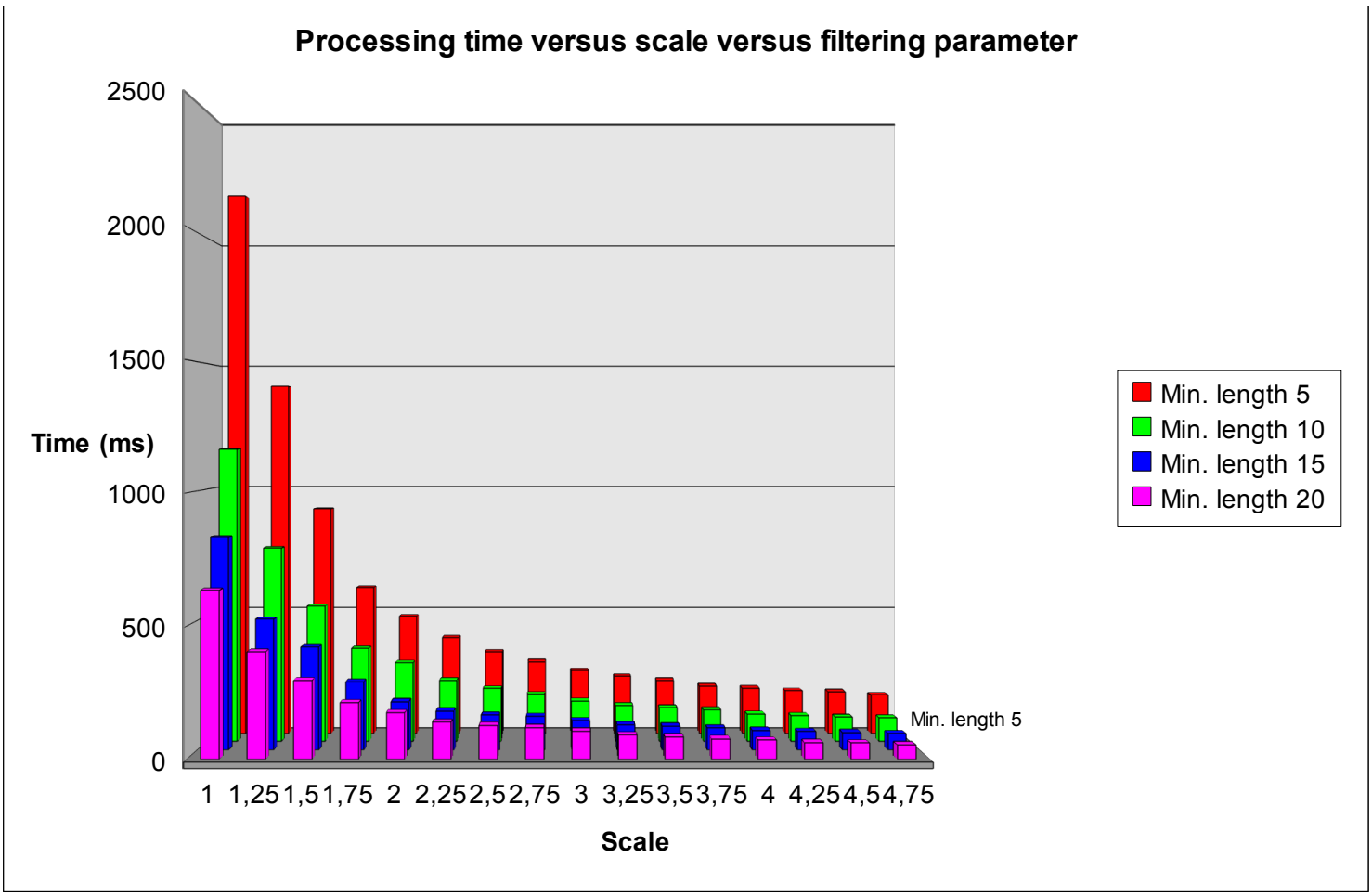


Figure 26. Processing time versus scale versus filtering parameter

The time that is spent since the map is requested from the client, until the information is stored locally depends on three steps: filtering process in the server, sending of the file, and loading (i.e. the XML parsing and storing) of the map in the client. Table 12 shows the time spent in all three steps and the total time since the map is requested until it is ready to display. All measurements were made over 10 executions. The sending time was taken using the KTHOPEN wireless LAN in Electrum building. This time was calculated since the client sends the URL to the server (there was no DNS lookup to get the address of the web server), until the complete SVG filtered map is received. From this time we have subtracted the time spent by the server to generate the SVG file after receiving the request from the client, hence the resulting time is only the one spent in communication.

Table 12. Load times in the client

Minimum length (user units)	Filtering time (seconds)	Sending Time (seconds)	Parsing Time (seconds)	Total (seconds)
5	3.033	6.214	35.194	44.411
10	3.007	3.412	15.283	21.702
15	2.980	3.079	12.122	18.181
20	2.853	2.972	8.968	14.793

Source code for the client and for the web service that generates the SVG filtered map are included as Appendix C.

Figure 27. HP iPAQ 5550 displaying the sample map

We can divide the time that the client application takes to display a portion of the map in two groups: the time that is spent in the iteration and transformations of the lines to display, and the time that is spent in drawing the lines in the bitmap displayed on the screen. Table 13 shows a comparison between the total times obtained in the measurements of Table 11, and the times corresponding to the drawing of the lines, for a number of scale values and filtering parameters of 5, 10, and 15. The lines are drawn using the *DrawLine* method of the class Graphics in .NET Framework, that "Draws a line connecting the two points specified by the coordinate pairs". The time spent drawing the lines represents an important percentage from the total time, specially when the scale factor and the filtering parameter are low (more lines have to be displayed). When small amounts of lines has to be drawn (larger scale values), the time spent iterating in the lines and performing calculations gains importance regarding to the drawing time, as the number of lines visited in the iteration stays the same, but less are

displayed.

Table 13. Drawing time statistics

	Min. Lenght = 5			Min. Lenght = 10			Min. Lenght = 15		
Scale	**Total time (ms)**	**Drawing time (ms)**	**% from total**	**Total time (ms)**	**Drawing time (ms)**	**% from total**	**Total time (ms)**	**Drawing time (ms)**	**% from total**
1	2190	1805	**82.42%**	1160	945	**81.47%**	823	659	**80.07%**
1.25	1413	1103	**78.06%**	768	604	**78.65%**	505	390	**77.23%**
1.5	914	661	**72.32%**	537	399	**74.30%**	398	298	**74.87%**
1.75	594	398	**67.00%**	369	263	**71.27%**	264	189	**71.59%**
2	477	299	**62.68%**	313	217	**69.33%**	187	118	**63.10%**
2.25	391	234	**59.85%**	243	152	**62.55%**	151	85	**56.29%**
2.5	332	183	**55.12%**	210	124	**59.05%**	135	76	**56.30%**
2.75	292	157	**53.77%**	187	109	**58.29%**	130	73	**56.15%**
3	256	126	**49.22%**	161	86	**53.42%**	114	59	**51.75%**
3.25	232	112	**48.28%**	143	69	**48.25%**	98	47	**47.96%**
3.5	216	99	**45.83%**	133	67	**50.38%**	92	43	**46.74%**
3.75	192	76	**39.58%**	125	59	**47.2%**	85	35	**41.18%**
4	184	68	**36.58%**	109	45	**41.28%**	76	27	**35.53%**
4.25	174	60	**34.48%**	103	37	**35.92%**	72	25	**34.72%**
4.5	169	58	**34.32%**	96	35	**36.92%**	68	16	**23.53%**
4.75	156	48	**30.77%**	94	32	**34.04%**	63	12	**19.05%**

5 Conclusion and future work

The challenge of utilizing sensors to design an application that is aware of the user's context was the core problem to be discussed in this project. The NXP Semiconductors KMZ52 that measures magnetic fields was the selected sensor to implement an application that allows the user of a personal digital assistant to visualize a map adapted to the device's orientation. Interfacing the sensor to the device means powering the sensor by means of the own device, and being able to read the values about earth's magnetic field offered by the sensor. This interface was theoretically proved to be feasible in this thesis project, as the device's serial lines were experimentally determined to provide enough power supply to feed the components needed (MSP430F2618 microcontroller, MAX1039 ACD, level shifting chip MAX3241, and KMZ52 sensor), and the serial communication was successful between the device and the microcontroller. The achievement of a real interface was skipped in order to focus the project on the implementation of the device's application.

The development of an application that displays a map according to the device's position and orientation, and the zoom level selected by the user, led to the question of how to distribute the computation between the client and the server. Since we have a server that stores a number a files representing different maps, and a device that request some area of a certain map with a given orientation and zoom level, different approaches can be analyzed. The maps are stored in a vector graphics format (SVG) to allow transformations without losing picture quality and avoid having files of large size. In the first approach attempted, the computation load was placed completely on the server. In this solution the client requests the area to display with the corresponding parameters, and the server renders the SVG file to a PNG image rotated accordingly to the orientation of the device and send it to the client. In this approach the client only has to request a map, receive the map as an image, and display this image. Two different rendering programs for SVG files were tried, and two different ways of rotating the image (modifying the SVG XML code before rendering, and rotating the PNG file after rendering) were implemented, but the minimum response time obtained was in all cases too excessive for the response requirements of the application. This lead to a more client-oriented approach, where the SVG rendering is performed in the PDA. Due to the lack of SVG viewers and libraries that could allow us to implement the application in the HP IPAQ, we decided to develop our own (minimal) SVG viewer. With this solution, once the SVG file is rendered in the client, the display of an area of the map and its corresponding transformations is relatively fast, but we found that the rendering process is still excessively slow for large SVG files. This led to a web service that filters the SVG file *before* sending it to the client, to remove irrelevant objects and tiny lines that are not necessary for the required level of detail in the client application. The client can specify the minimum length of the lines that are to be displayed, and the server removes lines smaller than this value from the SVG file. The result is a new file that can be parsed faster (as a function of the filtering parameter). This parameter also has an influence on the time required by the client to display a certain

area of the map, as a higher value of the filter parameter leads to fewer lines to display. With this solution we got fast responses (of the order of several hundreds of milliseconds) when the application is running with certain zoom level and without the entire map being shown, while still providing an acceptable quality of image.

An additional improvement in performance would have the server perform the transformations on the map and only send to the client the lines that are going to appear on the screen. However, the process of drawing the lines on a bitmap was generally more cost than performing the transformations on the points. Hence there is no point in having the server calculate the transformations, as the lines have to be drawn at the client anyway (this task occupies most of the execution time), and sending and parsing the SVG file from the server has to be done every time there is a change in location, scaling, or orientation.

5.1 Future work

The near term future work includes the following:

- **Interface between the sensor and the PDA**: This involves the design of the circuitry to connect the KMZ52 to the WASA board in order to be able to make readings of the sensor from the microcontroller in the board. For this purpose I^2C communication has to be achieved between the microcontroller and the ADC chip where the outputs of the sensor would be digitized. A more ambitious effort would be to design the circuit powered by the PDA serial lines, only using the necessary components - leading to a minimal part count and minimal power solution.
- **Analysis of the efficiency and image quality compromise:** When choosing the minimum length parameter to filter a map, an appropriate compromise between image quality and efficiency has to be achieved for every picture. Different pictures can have different optimal filtering values, but an study with a number of building maps could be done to evaluate if there is a way to estimate a good value for this parameter.
- **Include other SVG features**: Development of some features of SVG that were left out of our limited implementation, such as colors, lines of different widths, and curves in the *path* command.
- **Experiments with more sample files:** In this thesis all the experiments were performed using the same sample SVG file. This was necessary to compare multiple different alternative solutions, but more files should be tried to develop greater insight into what is the most suitable way to implement the desired functionality.

References

[1] **A.K. Dey** *Understanding and Using* Context. Personal and Ubiquitous Computing Journal, Vol. 5 (1), February 2001, pp. 4-7

[2] **Albrecht Schmidt**. *Ubiquitous Computing – Computing in Context.* Ph.D. Thesis. Lancaster University, Computing Department. November, 2002. http://www.comp.lancs.ac.uk/~albrecht/phd/

[3] **Wi-Fi RTLS and Site Survey Solutions [last accessed February 22, 2010]** http://www.ekahau.com/

[4] **Haruumi Shiode.** *In-building Location Sensing Based on WLAN Signal Strenght: Realizing a Presence User Agent, Master Thesis.* Stockholm : Royal Institute of Technology (KTH), School of Information and Communication, 2008. COS/CCS 2008-04, http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/080314-Haruumi_Shiode-with-cover.pdf

[5] **H.W.P. Beadle, G.Q. Maguire Jr., and M.T. Smith.** *Smart Badge: It beeps, It flashes, It knows when you are hot and sweaty*, (Submitted to) IEEE International Symposium on Wearable Computing, Cambridge, MA, USA, Oct. 1997, http://web.it.kth.se/~maguire/LocationAware/wearable2/wearable2.html

[6] **Henry Sinnreich and Alan B Johnston.** *Internet communications using SIP: delivering VoIP and multimedia services with session initiation protocol.* 2nd Edition, Wiley, August 2006, ISBN: 0-471-77657-2.

[7] Infrared Data Association [last accessed February 22, 2010] http://www.irda.org/

[8] **Bemnet Tesfaye Merha.** *Secure Context-Aware Mobile SIP User Agent, Master Thesis.* Stockholm : Royal Institute of Technology (KTH), School of Information and Communication, July 2009. http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/090705-Bemnet_Tesfaye_Merha-with-cover.pdf

[9] Mark T. Smith and Gerald Q. Maguire Jr. SmartBadge/BadgePad III , HP Labs and Royal Institute of Technology (KTH) web page [last accessed February 22, 2010] http://web.it.kth.se/~maguire/badge3/badge3.html

[10] Mark T. Smith and Gerald Q. Maguire Jr. SmartBadge/BadgePad version 4, HP Labs and Royal Institute of Technology (KTH) web page, last modified 27 June 2006, page initially created in 2003 [last accessed February 22, 2010]
http://www.it.kth.se/~maguire/badge4.html

[11] **Mat Hans, April Slayden, Mark Smith, Banny Banerjee, and Arvind Gupta.** *DJammer: A New Digital, Mobile, Virtual, Personal Musical Instrument*, Technical Report HPL-2005-81 , Hewlett-Packard Laboratories, Palo Alto, California · May 5, 2005.

[12] **April Slayden, Mirjana Spasojevic, Mat Hans, and Mark T. Smith.** *The DJammer: 'Air-Scratching' and Freeing the DJ to Join the Party*, presented as an interactive poster as part of the Late Breaking Results Technical Program, ACM CHI (Conference on Human Factors in Computing Systems) 2005, Portland, Oregon · April 2-7, 2005. Available as Technical Report HPL-2005-26, Hewlett-Packard Laboratories, Palo Alto, California · March 19, 2005.

[13] **Mat Hans and Mark Smith.** *Interacting with Audio Streams for Entertainment and Communication*, ACM Multimedia, Berkeley, California, November 2003.

[14] **Mat Hans and Mark Smith.** *A Wearable Networked MP3 Player and "Turntable" for Collaborative Scratching*, IEEE International Symposium on Wearable Computers, White Plains, New York, October 2003.

[15] **Thor Hådén.** *IPv6 Home Automation, Bachelor Thesis.* Stockholm: Royal Institute of Technology (KTH), School of Information and Communication, June 2009. http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/090601-Thor_Haaden.pdf

[16] **Daniel Hübinette.** *Occupancy Sensor System: For Context-aware Computing*, Masters thesis, Department of Communication Systems, Royal Institute of Technology (KTH), COS/CCS 2007-26, December 2007 http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/071221-Daniel_Hubinette_Master_Thesis-with-cover.pdf

[17] **Xueliang Ren,** *A Meeting Detector to Provide Context to a SIP Proxy*, Masters thesis, Department of Communication Systems, Royal Institute of Technology (KTH), COS/CCS 2008-24, October 2008 http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/081025-Xueliang_Ren-with-cover.pdf

[18] **Darwin Valderas Núñez.** *Integration of sensor nodes with IMS*, Masters thesis, Department of Communication Systems, Royal Institute of Technology (KTH), COS/CCS 2008-22, October 2008. http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/081008-DarwinValderas-with-cover.pdf

[19] NOAA's Geophysical Data Center – Geomagnetic Data [last accessed February 22, 2010]

http://www.ngdc.noaa.gov/geomagmodels/Declination.jsp

[20] FT232R – USB UART [last accessed February 22, 2010]

http://www.ftdichip.com/Products/FT232R.htm

[21] IAR Embedded Workbench Kickstart – Free IDE [last accessed February 22, 2010]

http://focus.ti.com/docs/toolsw/folders/print/iar-kickstart.html

[22] MAX3241E [last accessed February 22, 2010]

http://www.maxim-ic.com/quick_view2.cfm/qv_pk/1780

[23] Inkscape [last accessed February 22, 2010]

http://www.inkscape.org

[24] Batik SVG Rasterizer [last accessed February 22, 2010]

http://xmlgraphics.apache.org/batik/tools/rasterizer.html

[25] PHP GD library [last accessed February 22, 2010]

http://php.net/manual/en/book.image.php

[26] Ikivo [last accessed February 22, 2010]

http://www.ikivo.com

[27] Bitflash [last accessed February 22, 2010]

http://www.bitflash.com/

[28] Intesis [last accessed February 22, 2010]

http://www.intesis.com/eng/entrada_frame_eng.htm

[29] Paths – SVG 1.1 [last accessed February 22, 2010]

http://www.w3.org/TR/SVG/paths.html

[30] Coordinate Systems, Transformations and Units – SVG [last accessed February 28, 2010]

http://www.w3.org/TR/SVG/coords.html#Units

[31] TFA-Dostmann [last accessed March 04, 2010]

http://www.tfa-dostmann.de

Appendix A: Changes to WASA board code to support the second UART

Function to write in the transmit pin of UART1

```
void mts_putchar(char a_char_to_print)
{
  while (!(UC1IFG&UCA1TXIFG));   // wait until USCI_A1 TX buffer empty
  UCA1TXBUF = a_char_to_print;
}
```

Function to set up UART0 and UART1

```
void set_up_UART_A0_and_UART_A1( void )
{
  /* Now, set up UART A0.  It defaults to 115200 baud.  It uses XT2
    for the source of the baud clock.
*/
  P3SEL = 0xf0;                 // Use P3.4,5,6,7 for UART A0, A1 TXD and RXD
  BCSCTL2 |= SELS;              // SMCLK now sourced from XT2
  UCA0CTL1 |= UCSSEL_2;         // UART clock now sourced from SMCLK

  //Added by me
  UCA1CTL1 |= UCSSEL_2;

// Set up the baud divisors
  UCA0BR0 = 138;                  // 16 Mhz giving 115200 baud
  UCA0BR1 = 0;                    // 16 Mhz giving 115200 baud
  UCA0MCTL = UCBRS2 + UCBRS1 + UCBRS0;  // Modulation UCBRSx = 7

  //Added by me
  UCA1BR0 = 138;                  // 16 Mhz giving 115200 baud
  UCA1BR1 = 0;                    // 16 Mhz giving 115200 baud
  UCA1MCTL = UCBRS2 + UCBRS1 + UCBRS0;  // Modulation UCBRSx = 7

  UCA0CTL1 &= ~UCSWRST;         // Start the UART
  IE2 |= UCA0RXIE;              // And enable UART A0 RX interrupt

  //Added by me
  UCA1CTL1 &= ~UCSWRST;         // Start the UART
  UC1IE |= UCA1RXIE;            // And enable UART A0 RX interrupt

  __bis_SR_register(GIE);       // And enable interupts
}
```

Interrupt routine for UART1 reception

```
// For UART1 receive
#pragma vector=USCIAB1RX_VECTOR
__interrupt void USCI1RX_ISR(void)
{
  UC1IE &= ~UCA1RXIE;                 // disable UART A0 RX interrupt
  input_queue[queue_in_ptr] = UCA1RXBUF;  //put character in queue

// Only echo a character if the echo flag is on.  The echo flag
// can be set or cleared using the E1 or E0 AT command.
// It defaults cleared (no echo).

  if(basic_at_command_flags & flag_e)
    mts_putchar(input_queue[queue_in_ptr]);  // echo character

  ++queue_in_ptr;                   // increment the pointer
  _BIC_SR_IRQ(LPM0_bits);           // Clear CPUOFF bit from 0(SR)
  UC1IE |= UCA1RXIE;                // And enable UART A0 RX interrupt
  __bis_SR_register(GIE);           // And enable interupts
}
```

Appendix B: Server-Oriented Map display: PDA and server applications

PDA application's source code (C#)

```
HttpWebRequest request = (HttpWebRequest)
  WebRequest.Create("http://192.168.2.2:85/mapdisplay.php?a=" + angle.Value.ToString() +
  "&x=" + x.Value.ToString() + "&y=" + y.Value.ToString() + "&tam=" + tam.Value.ToString());

Stream stream = request.GetResponse().GetResponseStream();
Bitmap img = new Bitmap(stream);

map.Image = img;

request.Abort();
```

Web service's source code for rotation of the PNG image (PHP)

```
<?php

function CutImage($imgOriginal, $ImgDest, $Width, $Heigth, $x, $y){

   $imgFinal = imagecreatetruecolor($Width, $Heigth);
   imagecopyresampled($imgFinal, $imgOriginal, 0, 0, $x, $y, $Width, $Heigth, $Width, $Heigth);
   imagepng($imgFinal, $ImgDest, 9);
   imagedestroy($imgFinal);
}

$angle=$_GET["a"];
$x=$_GET["x"];
$y=$_GET["y"];
$tam=$_GET["tam"];

$frameTam=250;
$frameBigTam=375;


exec("Inkscape\inkscape.exe -f ..\map.svg -e ..\map.png -a ".($x-intval($tam*0.75)).":".($y-
intval($tam*0.75)).":".($x+intval($tam*0.75)).":".($y+intval($tam*0.75))." -w".$frameBigTam." -h".
$frameBigTam." -y 255");


$image = 'map.png';

$source = imagecreatefrompng($image);
```

```
$rotate = imagerotate($source, $angle, 0);


$newTam=sin(deg2rad($angle%90))*$frameBigTam + cos(deg2rad($angle%90))*$frameBigTam;

CutImage($rotate,'mapfinal.png',$frameTam,$frameTam,($newTam/2)-($frameTam/2),($newTam/2)-
($frameTam/2));


header("Content-type: application/octet-stream");
header("Content-Disposition: attachment; filename=\mapfinal.png\n");
$fp=fopen("mapfinal.png", "r");
fpassthru($fp);

?>
```

Web service's source code for rotation of the SVG image (PHP)

```
<?php

$angle=$_GET["a"];
$x=$_GET["x"];
$y=$_GET["y"];
$tam=$_GET["tam"];

$handle = @fopen("batik-1.7/Electrum1-plan3and4-_0001.svg", "r+");

fgets($handle);
fwrite($handle,"<g  transform=\"translate(".$x.",".$y."),  rotate(".$angle."),  translate(-".$x.",-".$y.")\">
");

fclose($handle);

exec("java -jar batik-1.7/batik-rasterizer.jar batik-1.7/Electrum1-plan3and4-_0001.svg -w 300 -h 300 -a
".($x-$tam/2).",".($y-$tam/2).",".$tam.",".$tam." -bg 255.255.255.255");

header("Content-type: application/octet-stream");
header("Content-Disposition: attachment; filename=\mapfinal.png\n");
$fp=fopen("batik-1.7/Electrum1-plan3and4-_0001.png", "r");
fpassthru($fp);

?>
```

Appendix C: Client-Oriented Map display: PDA and server applications

PDA Application: Class Map

```
class Map
  {
    public ArrayList lines; //array to store connected lines
    public ArrayList points; //array to store the points of a connected line
    public float[] cos = new float[360]; //array to store Cos values of every angle
    public float[] sin = new float[360]; //array to store Sin values of every angle

    public Map()
    {
       lines = new ArrayList();
       points = new ArrayList();
    }

    //this function precalculates the Sin and Cos for every integer angle
    public void fillSinCosVectors()
    {

       for (int i = 0; i < 360; i++)
       {
          sin[i] = (float)Math.Sin((double)i * (Math.PI / 180));
          cos[i] = (float)Math.Cos((double)i * (Math.PI / 180));
       }
    }

    //this function inserts one point (two coordinates written in a string)
    //in the current points array
    //if one of the coordinates is "-1" means that the values is the same that
    //in the last point stored (H and V commands in SVG)
    public void insertPoint(string point)
    {
       float x = 0;
       float y = 0;

       StringReader reader = new StringReader(point);
       string num = "";
       char[] c = new char[1];
       int state=0;
       while (reader.Read(c, 0, 1) != 0)
       {
          if (c[0] != ' ' && c[0] != ',')
             num = num + c[0];
          else
          {
             if (state == 0 && num != "")
             {
                x = float.Parse(num);
                if (x == -1)
                   x = ((FloatPoint)points[points.Count - 1]).X;
                state = 1;
                num = "";
             }
```

```
            else if (state == 1 && num != "")
            {
                y = float.Parse(num);
                num = "";
                if (y == -1)
                    y = ((FloatPoint)points[points.Count - 1]).Y;
                points.Add(new FloatPoint(x, y));
            }
        }

    }
    if (num != "")
    {
        y = float.Parse(num);
        if (y == -1)
            y = ((FloatPoint)points[points.Count - 1]).Y;
        points.Add(new FloatPoint(x, y));
    }
}

//this function store the current array of points in the lines array
//and create the new points array. It finishes the current connected line
//and start a new one
public void newLine()
{
    if (points.Count != 0)
    {
        lines.Add(points);
        points = new ArrayList();
    }
}

//it inserts a new point that is the first one in the line, to make a closepath
public void closePath()
{
    if (points.Count != 0)
        points.Add(points[0]);

}

//this function return a bitmap with the picture drawn, taking the arguments of
//position, scale, angle, width of the lines and size of the frame
public Bitmap setPicture(int size, int x, int y, float scaleFactor, int angle, float penWidth)
{
    Bitmap b = new Bitmap(size, size);
    Graphics g = Graphics.FromImage(b);
    float x1=0, x2, y1=0, y2, xTemp;
    float px1, px2, py1, py2;
    bool lastc;

    //convert the width of the lines according to the scale
    penWidth = penWidth * scaleFactor;

    Pen pen = new Pen(Color.Black, penWidth);
    g.Clear(Color.Cornsilk);

    g.DrawRectangle(pen, 0, 0, size, size);

    //calculation of the clipping frame
    int center = size / 2;
```

```
int clipFrame = (int)(center * ((float)1 / scaleFactor) * (sin[angle % 90] + cos[angle % 90]));
int minX = x - clipFrame;
int maxX = x + clipFrame;
int minY = y - clipFrame;
int maxY = y + clipFrame;

//as the angle does not change, we can store the result in a single variable
//to make the access faster
float cos_ = cos[angle];
float sin_ = sin[angle];

//iteration along the lines
for (int i = 0; i < lines.Count; i++)
{

   px1 = ((FloatPoint)((ArrayList)lines[i])[0]).X;
   py1 = ((FloatPoint)((ArrayList)lines[i])[0]).Y;

   lastc = false;

   //iteration along the points in a line
   for (int j = 1; j < ((ArrayList)lines[i]).Count; j++)
   {
      px2 = ((FloatPoint)((ArrayList)lines[i])[j]).X;
      py2 = ((FloatPoint)((ArrayList)lines[i])[j]).Y;

      //check if the line is inside the clipping frame
      if (((px1 > minX && px1 < maxX) && (py1 > minY && py1 < maxY)) ||
          ((px2 > minX && px2 < maxX) && (py2 > minY && py2 < maxY)))
      {
         //check if this point has been calculated in the previous iteration
         if (lastc == false)
         {
            x1 = px1 - x;
            y1 = py1 - y;
            xTemp = x1;

            x1 = (float)((double)((xTemp) * scaleFactor) * cos_ -
                  (double)((y1) * scaleFactor) * sin_) + center;
            y1 = (float)((double)((xTemp) * scaleFactor) * sin_ +
                  (double)((y1) * scaleFactor) * cos_) + center;


            if ((x1 - (int)x1) > 0.50)
               x1 = x1 + 1;
            if ((y1 - (int)y1) > 0.50)
               y1 = y1 + 1;
         }

         x2 = px2 - x;
         y2 = py2 - y;
         xTemp = x2;

         x2 = (float)((double)((xTemp) * scaleFactor) * cos_ -
               (double)((y2) * scaleFactor) * sin_) + center;
         y2 = (float)((double)((xTemp) * scaleFactor) * sin_ +
               (double)((y2) * scaleFactor) * cos_) + center;


         if ((x2 - (int)x2) > 0.50)
```

```
                x2 = x2 + 1;
            if ((y2 - (int)y2) > 0.50)
                y2 = y2 + 1;
            if (((x1 > 0 && x1 < size) && (y1 > 0 && y1 < size)) || ((x2 > 0 && x2 < size) && (y2 > 0
              && y2 < size)))
            {
                g.DrawLine(pen, (int)x1, (int)y1, (int)x2, (int)y2);
            }

            x1 = x2;
            y1 = y2;
            lastc = true;
          }
          else lastc = false;
          px1 = px2;
          py1 = py2;

        }
      }
      return b;
    }

  }
```

PDA Application: Class SVGParser

```
class SVGParser
  {

    Map map;

    public SVGParser(Map map)
    {
      this.map = map;
    }

    //this function read the XML file containing the map and gets every "d"
    //attribute in a "Path" element
    public void ReadSVG()
    {
      string d;
      string path =
        Path.GetDirectoryName(System.Reflection.Assembly.GetExecutingAssembly().GetName().CodeBase);

      XmlTextReader recXML = new XmlTextReader(path + @"\map.svg");

      recXML.WhitespaceHandling = WhitespaceHandling.None;
      recXML.MoveToContent();

      string prevNodeName = "";


      while (recXML.Read())
      {
        if (recXML.NodeType == XmlNodeType.Text)
        {
```

```
        }
        if (recXML.NodeType == XmlNodeType.Element)
        {
            prevNodeName = recXML.Name;
            if (prevNodeName == "path")
            {
                d = recXML.GetAttribute("d");
                saveLines(d);
            }
        }
        else
        {
            prevNodeName = "";
        }
    }

    recXML.Close();

}

//this function receives a string containing the "d" attribute of a "Path" element
//and parses this information to store it in the data structures
void saveLines(string d)
{
    StringReader reader = new StringReader(d);
    string point="";
    char[] c = new char[1];
    string state = "ini";

    while (reader.Read(c, 0, 1)!=0)
    {
        if (c[0] == 'M' || c[0] == 'm')
        {
            if (state != "ini" && state != "nothing")
            {
                if (state == "moveto")
                    map.newLine();
                else if (state == "horizontal")
                    point = point + " -1";
                else if (state == "vertical")
                    point = "-1 " + point;
                map.insertPoint(point);
            }
            point = "";
            state = "moveto";
        }
        else if (c[0] == 'L' || c[0] == 'l')
        {
            if (state != "ini" && state != "nothing")
            {
                if (state == "moveto")
                    map.newLine();
                else if (state == "horizontal")
                    point = point + " -1";
                else if (state == "vertical")
                    point = "-1 " + point;
                map.insertPoint(point);
            }
            point = "";
            state = "lineto";
```

```
}
else if (c[0] == 'H' || c[0] == 'h')
{
   if (state != "ini" && state != "nothing")
   {
      if (state == "moveto")
         map.newLine();
      else if (state == "horizontal")
         point = point + " -1";
      else if (state == "vertical")
         point = "-1 " + point;
      map.insertPoint(point);
   }
   point = "";
   state = "horizontal";
}
else if (c[0] == 'V' || c[0] == 'v')
{
   if (state != "ini" && state != "nothing")
   {
      if (state == "moveto")
         map.newLine();
      else if (state == "horizontal")
         point = point + " -1";
      else if (state == "vertical")
         point = "-1 " + point;
      map.insertPoint(point);
   }
   point = "";
   state = "vertical";
}
else if (c[0] == 'Z' || c[0] == 'z')
{
   if (state != "ini" && state != "nothing")
   {
      if (state == "moveto")
         map.newLine();
      else if (state == "horizontal")
         point = point + " -1";
      else if (state == "vertical")
         point = "-1 " + point;
      map.insertPoint(point);
      map.closePath();
   }
   point = "";
   state = "nothing";
}
else if (c[0] >= 'A' && c[0] <= 'Z' && (c[0] != 'L' || c[0] != 'l') && (c[0] != 'M' || c[0] != 'm') &&
        (c[0] != 'Z' || c[0] != 'z'))
{
   if (state != "ini" && state != "nothing")
   {
      if (state == "moveto")
         map.newLine();
      if (state == "horizontal")
         point = point + " -1";
      else if (state == "vertical")
         point = "-1 " + point;
      map.newLine();
      map.insertPoint(point);
```

```
                }
                point = "";
                state = "nothing";
            }
            else
            {
                point = point + c[0];
            }
        }

        if (state == "moveto")
            map.newLine();
        else if (state == "horizontal")
            point = point + " -1";
        else if (state == "vertical")
            point = "-1 " + point;
        if (state != "ini" && state != "nothing")
        {
            map.insertPoint(point);
        }
        map.newLine();
    }
}
```

PDA Application: Class FloatPoint

```
class FloatPoint
{
    public float X;
    public float Y;

    public FloatPoint(float X, float Y)
    {
        this.X = X;
        this.Y = Y;
    }
}
```

PDA Application: Class Display

```
public partial class Display : Form
{
    Map map;
    int size; //size of the box where the map is displayed
    float penWidth; //width of the lines of the map
    float scale; //scale factor of the map
    int x,y; //(x,y) current location
    int angle; //current angle of orientation

    public Display()
    {
        size = 600;
        penWidth = (float)2.5;
        map = new Map();
        scale = 1;
        x = 350;
```

```
    y = 350;
    angle = 0;

    InitializeComponent();

    //parse the SVG file and store the data in an instance of the class Map
    SVGParser s=new SVGParser(map);
    s.ReadSVG();
    //precalculation of Sin and Cos functions
    map.fillSinCosVectors();
    //display map
    pictureBox1.Image = map.setPicture(size, x, y, scale, angle, penWidth);
}


//increase the scaleFactor and draw the picture
private void zoomIn_Click(object sender, EventArgs e)
{
    pictureBox1.Image.Dispose();
    scale += (float)0.1;
    pictureBox1.Image = map.setPicture(size, x, y, scale, angle, penWidth);
}

//decrease the scaleFactor and draw the picture
private void zoomOut_Click(object sender, EventArgs e)
{
    pictureBox1.Image.Dispose();
    scale -= (float)0.1;
    pictureBox1.Image = map.setPicture(size, x, y, scale, angle, penWidth);
}

//change current position and draw the picture
private void up_Click(object sender, EventArgs e)
{
    pictureBox1.Image.Dispose();
    y -= 20;
    pictureBox1.Image = map.setPicture(size, x, y, scale, angle, penWidth);
}

//change current position and draw the picture
private void down_Click(object sender, EventArgs e)
{
    pictureBox1.Image.Dispose();
    y += 20;
    pictureBox1.Image = map.setPicture(size, x, y, scale, angle, penWidth);
}

//change current position and draw the picture
private void right_Click(object sender, EventArgs e)
{
    pictureBox1.Image.Dispose();
    x += 20;
    pictureBox1.Image = map.setPicture(size, x, y, scale, angle, penWidth);
}

//change current position and draw the picture
private void left_Click(object sender, EventArgs e)
{
    pictureBox1.Image.Dispose();
    x -= 20;
```

```
            pictureBox1.Image = map.setPicture(size, x, y, scale, angle, penWidth);
        }

        //change orientation and draw the picture
        private void clockWise_Click(object sender, EventArgs e)
        {
            pictureBox1.Image.Dispose();
            angle += 5;
            if (angle == 360) angle = 0;
            pictureBox1.Image = map.setPicture(size, x, y, scale, angle, penWidth);
        }

        //change orientation and draw the picture
        private void counterClockWise_Click(object sender, EventArgs e)
        {
            pictureBox1.Image.Dispose();
            angle -= 5;
            if (angle < 0) angle = 355;
            pictureBox1.Image = map.setPicture(size, x, y, scale, angle, penWidth);
        }
    }
```

Web Service source code for filtering an SVG file according to a given filtering value

```
<?php

include 'PHP_script_timer.php';

$timing_loops = 1;
for ($current_loop = 0; $current_loop < $timing_loops; $current_loop++) {
$time_start = microtime_float();


if (file_exists('map.svg')) {
    $xml = simplexml_load_file('map.svg');
}
else echo "No existe\n";

$info = $xml->xpath('//svg');
$width = $info[0]['width'];
$height = $info[0]['height'];
$precision=$_GET["minLenght"];



$xml2 = new XmlWriter();

$xml2->openURI('newMap.svg');

$xml2->startElement('svg');
$xml2->writeAttribute('xmlns:svg','http://www.w3.org/2000/svg');
$xml2->writeAttribute('xmlns','http://www.w3.org/2000/svg');
$xml2->writeAttribute('width',$width);
$xml2->writeAttribute('height',$height);
$xml2->startElement('g');
```

```
$result = $xml->xpath('//svg/g/g/g/path');

if($result){

foreach($result as $node){

 $state = "ini";
 $x = "";
 $y = "";
 $d = "";

 $lastx=0;
 $lasty=0;

 $points = (string)$node['d'];
 $tam = strlen($points);

 for($i=0;$i<$tam;$i++){


  if ($points[$i] == 'M' || $points[$i] == 'm')
  {
   $state="moveto";
   $i++;
   while($points[$i] != ' '){
    $x = $x.$points[$i];
    $i++;
   }
   while($points[$i] == ' ')
    $i++;
   while($points[$i] != ' ' && $points[$i]<'A' && $i<$tam){
    $y = $y.$points[$i];
    $i++;
   }
   while($points[$i] == ' ')
    $i++;

   //echo "M".(float)$x."-".(float)$y."<br> \n";

   $lastx = (float)$x;
   $lasty = (float)$y;


   $i--;
   $x = "";
   $y = "";

   }


   else if ($points[$i] == 'L' || $points[$i] == 'l')
   {

   if($points[$i] == 'l')$lowercase=1;
   else $lowercase=0;

   $i++;
```

```
while($points[$i] != ' '){
 $x = $x.$points[$i];
 $i++;
}
while($points[$i] == ' ')
 $i++;
while($points[$i] != ' ' && $points[$i]<'A' && $i<$tam){
 $y = $y.$points[$i];
 $i++;
}
while($points[$i] == ' ')
 $i++;

if($lowercase==1){
 $x = (float)$lastx + (float)$x;
 $y = (float)$lasty + (float)$y;
}


if(abs((float)$x - $lastx) > $precision || abs((float)$y - $lasty) > $precision){

//echo "L".(float)$x."-".(float)$y."<br> \n";

 if($state=="moveto"){
  $d = $d."M ".$lastx." ".$lasty." L ".$x." ".$y." ";
 }
 else if($state=="lineto"){
  $d = $d."L ".$x." ".$y." ";
 }

$lastx = (float)$x;
$lasty = (float)$y;

$state="lineto";

}
//else echo "NO <br> \n";

$i--;
$x = "";
$y = "";

}

else if ($points[$i] == 'H' || $points[$i] == 'h')
{

if($points[$i] == 'h')$lowercase=1;
else $lowercase=0;

$i++;

while($points[$i] != ' ' && $points[$i]<'A' && $i<$tam){
 $x = $x.$points[$i];
 $i++;
}
while($points[$i] == ' ')
 $i++;

if($lowercase==1){
```

```
 $x = (float)$lastx + (float)$x;
}

if(abs((float)$x - $lastx) > $precision){

//echo "H".(float)$x."<br> \n";

 if($state=="moveto"){
  $d = $d."M ".$lastx." ".$lasty." H ".$x." ";
 }
 else if($state=="lineto"){
  $d = $d." H ".$x." ";
 }

$lastx = (float)$x;

$state = "lineto";

}
//else echo "NO <br> \n";

$i--;
$x = "";

}

else if ($points[$i] == 'V' || $points[$i] == 'v')
{

if($points[$i] == 'v')$lowercase=1;
else $lowercase=0;

$i++;

while($points[$i] != ' ' && $points[$i]<'A' && $i<$tam){
 $y = $y.$points[$i];
 $i++;
}
while($points[$i] == ' ')
 $i++;

if($lowercase==1){
 $y = (float)$lasty + (float)$y;
}

if(abs((float)$y - $lasty) > $precision){

//echo "V".(float)$y."<br> \n";

if($state=="moveto"){
  $d = $d."M ".$lastx." ".$lasty." V ".$y." ";
 }
 else if($state=="lineto"){
  $d = $d." V ".$y." ";
 }

$lasty = (float)$y;

$state = "lineto";
```

```
  }
  //else echo "NO <br> \n";

  $i--;
  $y = "";

  }

  else if ($points[$i] == 'Z' || $points[$i] == 'z')
  {
  // echo "Z <br> \n";
  }
 }

 if($d!=""){
  $xml2->startElement('path');
  $xml2->writeAttribute('d', $d);
  $xml2->writeAttribute('stroke', '#000000');
  $xml2->writeAttribute('fill', 'none');
  $xml2->endElement();
  }
 }
}

$xml2->endElement();
$xml2->endElement();

$timing[] = microtime_float() - $time_start;
}
timing($timing);

header("Content-type: application/octet-stream");
header("Content-Disposition: attachment; filename=\map.svg\n");
$fp=fopen("newMap.svg", "r");
fpassthru($fp)

?>
```

TRITA-ICT-EX-2010:36

www.kth.se